#ifndef CONTROL\_H

#define CONTROL\_H

#include "process.h"

#include "threads.h"

void handle\_signal(int signal);

void handle\_input(Process \*processes, int num\_processes, Thread \*threads, int num\_threads);

void switch\_color\_mode();

void kill\_process\_by\_pid(int pid);

void kill\_thread\_by\_tid(int tid);

void sort\_by\_criterion();

#endif /\* CONTROL\_H \*/

#include "control.h"

#include "display.h"

#include "input.h"

#include "sort.h"

#include "read.h"

#include "sysinfo.h"

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <signal.h>

#include <termios.h>

#include <fcntl.h>

#include "process.h"

#include "threads.h"

#define MAX\_PROC 1000

#define MAX\_THREADS 1000

Process processes[MAX\_PROC];

Thread threads[MAX\_THREADS];

int num\_processes = 0;

int num\_threads = 0;

int display\_mode = 0; // 0 - процессы, 1 - потоки

void handle\_signal(int signal) {

if (signal == SIGINT) {

printf("Программа завершена.\n");

exit(0);

}

}

void handle\_input(Process \*processes, int num\_processes, Thread \*threads, int num\_threads) {

if (kbhit()) {

char c = getchar();

switch (c) {

case 'z':

switch\_color\_mode();

break;

case 't':

display\_mode = 1;

break;

case 'p':

display\_mode = 0;

break;

case 'k': {

int id;

char prompt[50];

if (display\_mode == 0) {

sprintf(prompt, "Введите PID процесса для удаления: ");

} else {

sprintf(prompt, "Введите TID потока для удаления: ");

}

printf("%s", prompt);

scanf("%d", &id);

if (display\_mode == 0) {

kill\_process\_by\_pid(processes, num\_processes, id);

} else {

kill\_thread\_by\_tid(threads, num\_threads, id);

}

break;

}

case 'q':

handle\_signal(SIGINT);

break;

case 's': {

char criterion[50];

printf("Введите критерий для сортировки (pid/tid, user, priority, virtual\_memory, resident\_memory, cpu\_usage): ");

scanf("%s", criterion);

if (display\_mode == 0) {

if (strcmp(criterion, "pid") == 0) {

sort\_processes\_by\_pid(processes, num\_processes);

} else if (strcmp(criterion, "user") == 0) {

sort\_processes\_by\_user(processes, num\_processes);

} else if (strcmp(criterion, "priority") == 0) {

sort\_processes\_by\_priority(processes, num\_processes);

} else if (strcmp(criterion, "virtual\_memory") == 0) {

sort\_processes\_by\_virtual\_memory(processes, num\_processes);

} else if (strcmp(criterion, "resident\_memory") == 0) {

sort\_processes\_by\_resident\_memory(processes, num\_processes);

} else if (strcmp(criterion, "cpu\_usage") == 0) {

sort\_processes\_by\_cpu\_usage(processes, num\_processes);

} else {

printf("Неверный критерий сортировки.\n");

}

} else {

if (strcmp(criterion, "tid") == 0) {

sort\_threads\_by\_pid(threads, num\_threads);

} else if (strcmp(criterion, "user") == 0) {

sort\_threads\_by\_user(threads, num\_threads);

} else if (strcmp(criterion, "priority") == 0) {

sort\_threads\_by\_priority(threads, num\_threads);

} else if (strcmp(criterion, "virtual\_memory") == 0) {

sort\_threads\_by\_virtual\_memory(threads, num\_threads);

} else if (strcmp(criterion, "resident\_memory") == 0) {

sort\_threads\_by\_resident\_memory(threads, num\_threads);

} else if (strcmp(criterion, "cpu\_usage") == 0) {

sort\_threads\_by\_cpu\_usage(threads, num\_threads);

} else {

printf("Неверный критерий сортировки.\n");

}

}

break;

}

default:

break;

}

}

}

void switch\_color\_mode() {

static int color\_mode = 0;

color\_mode = !color\_mode;

if (color\_mode) {

printf("\033[1;37m"); // white on black

} else {

printf("\033[0;30m"); // black on white

}

printf("\n");

}

void kill\_process\_by\_pid(Process \*processes, int num\_processes, int pid) {

// find process with given PID and kill it

for (int i = 0; i < num\_processes; i++) {

if (processes[i].pid == pid) {

kill(pid, SIGKILL);

return;

}

}

printf("Процесс с указанным PID не найден.\n");

}

void kill\_thread\_by\_tid(Thread \*threads, int num\_threads, int tid) {

// find thread with given TID and kill it

for (int i = 0; i < num\_threads; i++) {

if (threads[i].tid == tid) {

pthread\_cancel(threads[i].tid);

return;

}

}

printf("Поток с указанным TID не найден.\n");

}

#include "display.h"

#include "sysinfo.h"

#include <stdio.h>

void display\_sysinfo(SysInfo \*sysinfo) {

printf("Общее количество процессов: %d\n", sysinfo->total\_processes);

printf("Количество активных процессов: %d\n", sysinfo->running\_processes);

printf("Общее количество потоков: %d\n", sysinfo->total\_threads);

printf("Количество пользователей: %d\n", sysinfo->num\_users);

printf("Средняя нагрузка на систему (за последние 1, 5 и 15 минут): %.2f, %.2f, %.2f\n", sysinfo->load\_avg[0], sysinfo->load\_avg[1], sysinfo->load\_avg[2]);

printf("Использование ЦП пользователем: %.2f%%\n", sysinfo->cpu\_usage\_user);

printf("Использование ЦП системой: %.2f%%\n", sysinfo->cpu\_usage\_system);

}

void display\_processes(Process \*processes, int num\_processes) {

printf("%-8s %-15s %-8s %-12s %-12s %-8s %-8s %s\n",

"PID", "USER", "PRIORITY", "VM(KB)", "RM(KB)", "CPU(%)", "COMMAND");

for (int i = 0; i < num\_processes; ++i) {

printf("%-8d %-15s %-8d %-12ld %-12ld %-8.2f %-8s\n",

processes[i].pid, processes[i].user, processes[i].priority,

processes[i].virtual\_memory, processes[i].resident\_memory,

processes[i].cpu\_usage, processes[i].command);

}

}

void display\_threads(Thread \*threads, int num\_threads) {

printf("%-8s %-15s %-8s %-12s %-12s %-8s %-8s %s\n",

"PID", "USER", "TID", "VM(KB)", "RM(KB)", "CPU(%)", "COMMAND");

for (int i = 0; i < num\_threads; ++i) {

printf("%-8d %-15s %-8d %-12ld %-12ld %-8.2f %-8s\n",

threads[i].pid, threads[i].user, threads[i].tid,

threads[i].virtual\_memory, threads[i].resident\_memory,

threads[i].cpu\_usage, threads[i].command);

}

}

#ifndef DISPLAY\_H

#define DISPLAY\_H

#include "process.h"

void display\_sysinfo(SysInfo \*sysinfo);

void display\_processes(Process \*processes, int num\_processes);

void display\_threads(Thread \*threads, int num\_threads);

#endif /\* DISPLAY\_H \*/

#include "read.h"

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <dirent.h>

#include <unistd.h>

#include <errno.h>

void read\_sysinfo(SysInfo \*sysinfo) {

struct sysinfo info;

if (sysinfo\_mem\_info(&info) == 0) {

sysinfo->total\_processes = info.procs;

sysinfo->running\_processes = info.procs\_running;

sysinfo->total\_threads = info.totalhigh;

sysinfo->num\_users = info.users;

sysinfo->load\_avg[0] = info.loads[0];

sysinfo->load\_avg[1] = info.loads[1];

sysinfo->load\_avg[2] = info.loads[2];

sysinfo->cpu\_usage\_user = info.totalcpu \* 100.0 / (1 << SI\_CPU\_SHIFT);

sysinfo->cpu\_usage\_system = info.totalsystemcpu \* 100.0 / (1 << SI\_CPU\_SHIFT);

} else {

printf("Ошибка чтения информации о системе\n");

}

}

int read\_processes(Process \*processes) {

DIR \*dir;

struct dirent \*entry;

int count = 0;

if ((dir = opendir("/proc")) == NULL) {

perror("Ошибка opendir()");

return -1;

}

while ((entry = readdir(dir)) != NULL && count < MAX\_PROC) {

if (isdigit(\*entry->d\_name)) {

char path[256];

sprintf(path, "/proc/%s/status", entry->d\_name);

FILE \*fp;

if ((fp = fopen(path, "r")) != NULL) {

char line[256];

while (fgets(line, sizeof(line), fp)) {

if (strncmp(line, "Pid:", 4) == 0) {

sscanf(line, "%\*s %d", &processes[count].pid);

} else if (strncmp(line, "Uid:", 4) == 0) {

sscanf(line, "%\*s %s", processes[count].user);

} else if (strncmp(line, "VmSize:", 7) == 0) {

sscanf(line, "%\*s %ld", &processes[count].virtual\_memory);

} else if (strncmp(line, "VmRSS:", 6) == 0) {

sscanf(line, "%\*s %ld", &processes[count].resident\_memory);

} else if (strncmp(line, "Cpu(s):", 7) == 0) {

sscanf(line, "%\*s %lf", &processes[count].cpu\_usage);

} else if (strncmp(line, "Cmdline:", 8) == 0) {

sscanf(line, "%\*s %[^\n]", processes[count].command);

}

}

fclose(fp);

count++;

}

}

}

closedir(dir);

return count;

}

int read\_threads(Thread \*threads) {

DIR \*dir;

struct dirent \*entry;

int count = 0;

if ((dir = opendir("/proc")) == NULL) {

perror("Ошибка opendir()");

return -1;

}

while ((entry = readdir(dir)) != NULL && count < MAX\_THREADS) {

if (isdigit(\*entry->d\_name)) {

char path[256];

sprintf(path, "/proc/%s/task/%s/status", entry->d\_name, entry->d\_name);

FILE \*fp;

if ((fp = fopen(path, "r")) != NULL) {

char line[256];

while (fgets(line, sizeof(line), fp)) {

if (strncmp(line, "Pid:", 4) == 0) {

sscanf(line, "%\*s %d", &threads[count].pid);

} else if (strncmp(line, "Tid:", 4) == 0) {

sscanf(line, "%\*s %d", &threads[count].tid);

} else if (strncmp(line, "Uid:", 4) == 0) {

sscanf(line, "%\*s %s", threads[count].user);

} else if (strncmp(line, "VmSize:", 7) == 0) {

sscanf(line, "%\*s %ld", &threads[count].virtual\_memory);

} else if (strncmp(line, "VmRSS:", 6) == 0) {

sscanf(line, "%\*s %ld", &threads[count].resident\_memory);

} else if (strncmp(line, "Cpu(s):", 7) == 0) {

sscanf(line, "%\*s %lf", &threads[count].cpu\_usage);

} else if (strncmp(line, "Cmdline:", 8) == 0) {

sscanf(line, "%\*s %[^\n]", threads[count].command);

}

}

fclose(fp);

count++;

}

}

}

closedir(dir);

return count;

}

#include "control.h"

#include "display.h"

#include "sysinfo.h"

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <signal.h>

int main() {

// Установка обработчика сигнала SIGINT (Ctrl+C)

signal(SIGINT, handle\_signal);

SysInfo sysinfo;

// Чтение данных о системе

read\_sysinfo(&sysinfo);

// Чтение данных при запуске

num\_processes = read\_processes(processes);

if (num\_processes < 0) {

fprintf(stderr, "Ошибка чтения процессов\n");

return 1;

}

num\_threads = read\_threads(threads);

if (num\_threads < 0) {

fprintf(stderr, "Ошибка чтения потоков\n");

return 1;

}

// Бесконечный цикл обновления данных

while (1) {

// Очистка экрана перед обновлением информации

system("clear");

// Отображение информации о системе

display\_sysinfo(&sysinfo);

// Чтение данных о процессах и потоках

num\_processes = read\_processes(processes);

if (num\_processes < 0) {

fprintf(stderr, "Ошибка чтения процессов\n");

return 1;

}

num\_threads = read\_threads(threads);

if (num\_threads < 0) {

fprintf(stderr, "Ошибка чтения потоков\n");

return 1;

}

// Отображение информации о процессах или потоках в зависимости от выбранного режима

if (display\_mode == 0) {

display\_processes(processes, num\_processes);

} else {

display\_threads(threads, num\_threads);

}

// Ожидание 1 секунда перед следующим обновлением данных

sleep(1);

// Обработка нажатия клавиш

handle\_input(processes, num\_processes, threads, num\_threads);

}

return 0;

}

#ifndef PROCESS\_H

#define PROCESS\_H

#define MAX\_PROC 1000

typedef struct {

int pid; // Идентификатор процесса

char user[50]; // Имя пользователя

int priority; // Приоритет процесса

long virtual\_memory; // Потребление виртуальной памяти (KB)

long resident\_memory; // Потребление физической памяти (KB)

float cpu\_usage; // % использования CPU

char command[100]; // Название команды, инициализировавшей процесс

} Process;

#endif /\* PROCESS\_H \*/

#ifndef READ\_H

#define READ\_H

#include "process.h"

#include "threads.h"

void read\_sysinfo(SysInfo \*sysinfo);

int read\_processes(Process \*processes);

int read\_threads(Thread \*threads);

#endif /\* READ\_H \*/

#include "sort.h"

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

int compare\_processes\_by\_pid(const void \*a, const void \*b) {

const Process \*process\_a = a;

const Process \*process\_b = b;

return process\_a->pid - process\_b->pid;

}

int compare\_processes\_by\_user(const void \*a, const void \*b) {

const Process \*process\_a = a;

const Process \*process\_b = b;

return strcmp(process\_a->user, process\_b->user);

}

int compare\_processes\_by\_priority(const void \*a, const void \*b) {

const Process \*process\_a = a;

const Process \*process\_b = b;

return process\_a->priority - process\_b->priority;

}

int compare\_processes\_by\_virtual\_memory(const void \*a, const void \*b) {

const Process \*process\_a = a;

const Process \*process\_b = b;

return process\_a->virtual\_memory - process\_b->virtual\_memory;

}

int compare\_processes\_by\_resident\_memory(const void \*a, const void \*b) {

const Process \*process\_a = a;

const Process \*process\_b = b;

return process\_a->resident\_memory - process\_b->resident\_memory;

}

int compare\_processes\_by\_cpu\_usage(const void \*a, const void \*b) {

const Process \*process\_a = a;

const Process \*process\_b = b;

return (int)(process\_a->cpu\_usage - process\_b->cpu\_usage);

}

void sort\_processes\_by\_pid(Process \*processes, int num\_processes) {

qsort(processes, num\_processes, sizeof(Process), compare\_processes\_by\_pid);

}

void sort\_processes\_by\_user(Process \*processes, int num\_processes) {

qsort(processes, num\_processes, sizeof(Process), compare\_processes\_by\_user);

}

void sort\_processes\_by\_priority(Process \*processes, int num\_processes) {

qsort(processes, num\_processes, sizeof(Process), compare\_processes\_by\_priority);

}

void sort\_processes\_by\_virtual\_memory(Process \*processes, int num\_processes) {

qsort(processes, num\_processes, sizeof(Process), compare\_processes\_by\_virtual\_memory);

}

void sort\_processes\_by\_resident\_memory(Process \*processes, int num\_processes) {

qsort(processes, num\_processes, sizeof(Process), compare\_processes\_by\_resident\_memory);

}

void sort\_processes\_by\_cpu\_usage(Process \*processes, int num\_processes) {

qsort(processes, num\_processes, sizeof(Process), compare\_processes\_by\_cpu\_usage);

}

int compare\_threads\_by\_pid(const void \*a, const void \*b) {

const Thread \*thread\_a = a;

const Thread \*thread\_b = b;

return thread\_a->pid - thread\_b->pid;

}

int compare\_threads\_by\_user(const void \*a, const void \*b) {

const Thread \*thread\_a = a;

const Thread \*thread\_b = b;

return strcmp(thread\_a->user, thread\_b->user);

}

int compare\_threads\_by\_priority(const void \*a, const void \*b) {

const Thread \*thread\_a = a;

const Thread \*thread\_b = b;

return thread\_a->priority - thread\_b->priority;

}

int compare\_threads\_by\_virtual\_memory(const void \*a, const void \*b) {

const Thread \*thread\_a = a;

const Thread \*thread\_b = b;

return thread\_a->virtual\_memory - thread\_b->virtual\_memory;

}

int compare\_threads\_by\_resident\_memory(const void \*a, const void \*b) {

const Thread \*thread\_a = a;

const Thread \*thread\_b = b;

return thread\_a->resident\_memory - thread\_b->resident\_memory;

}

int compare\_threads\_by\_cpu\_usage(const void \*a, const void \*b) {

const Thread \*thread\_a = a;

const Thread \*thread\_b = b;

return (int)(thread\_a->cpu\_usage - thread\_b->cpu\_usage);

}

void sort\_threads\_by\_pid(Thread \*threads, int num\_threads) {

qsort(threads, num\_threads, sizeof(Thread), compare\_threads\_by\_pid);

}

void sort\_threads\_by\_user(Thread \*threads, int num\_threads) {

qsort(threads, num\_threads, sizeof(Thread), compare\_threads\_by\_user);

}

void sort\_threads\_by\_priority(Thread \*threads, int num\_threads) {

qsort(threads, num\_threads, sizeof(Thread), compare\_threads\_by\_priority);

}

void sort\_threads\_by\_virtual\_memory(Thread \*threads, int num\_threads) {

qsort(threads, num\_threads, sizeof(Thread), compare\_threads\_by\_virtual\_memory);

}

void sort\_threads\_by\_resident\_memory(Thread \*threads, int num\_threads) {

qsort(threads, num\_threads, sizeof(Thread), compare\_threads\_by\_resident\_memory);

}

void sort\_threads\_by\_cpu\_usage(Thread \*threads, int num\_threads) {

qsort(threads, num\_threads, sizeof(Thread), compare\_threads\_by\_cpu\_usage);

}

#ifndef SORT\_H

#define SORT\_H

#include "process.h"

#include "threads.h"

void sort\_processes\_by\_pid(Process \*processes, int num\_processes);

void sort\_processes\_by\_user(Process \*processes, int num\_processes);

void sort\_processes\_by\_priority(Process \*processes, int num\_processes);

void sort\_processes\_by\_virtual\_memory(Process \*processes, int num\_processes);

void sort\_processes\_by\_resident\_memory(Process \*processes, int num\_processes);

void sort\_processes\_by\_cpu\_usage(Process \*processes, int num\_processes);

void sort\_threads\_by\_pid(Thread \*threads, int num\_threads);

void sort\_threads\_by\_user(Thread \*threads, int num\_threads);

void sort\_threads\_by\_priority(Thread \*threads, int num\_threads);

void sort\_threads\_by\_virtual\_memory(Thread \*threads, int num\_threads);

void sort\_threads\_by\_resident\_memory(Thread \*threads, int num\_threads);

void sort\_threads\_by\_cpu\_usage(Thread \*threads, int num\_threads);

#endif /\* SORT\_H \*/

// sysinfo.h

#ifndef SYSINFO\_H

#define SYSINFO\_H

typedef struct {

int total\_processes; // общее количество процессов

int running\_processes; // количество активных процессов

int total\_threads; // общее количество потоков

int num\_users; // количество пользователей

float load\_avg[3]; // средняя нагрузка на систему за последние 1, 5 и 15 минут

float cpu\_usage\_user; // использование ЦП пользователем

float cpu\_usage\_system; // использование ЦП системой

} SysInfo;

#endif // SYSINFO\_H

#ifndef THREADS\_H

#define THREADS\_H

#define MAX\_THREADS 1000

typedef struct {

int pid; // Идентификатор процесса

int tid; // Идентификатор потока

char user[50]; // Имя пользователя

int priority; // Приоритет потока

long virtual\_memory; // Потребление виртуальной памяти (KB)

long resident\_memory; // Потребление физической памяти (KB)

float cpu\_usage; // % использования CPU

char command[100]; // Название команды, инициализировавшей процесс

} Thread;

#endif /\* THREADS\_H \*/