**НЕГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ ЧАСТНОЕ** **УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ** **«МОСКОВСКИЙ ФИНАНСОВО-ПРОМЫШЛЕННЫЙ УНИВЕРСИТЕТ** **“СИНЕРГИЯ”»**

|  |  |  |
| --- | --- | --- |
| **Факультет/Институт** |  | Информационные техеологии |
|  |  | (наименование факультета/ Института) |
| **Направление/специальность** |  | **Информационные системы и программирование** |
| **подготовки:** |  | (код и наименование направления /специальности подготовки) |
| **Форма обучения:** |  | Очная |
|  |  | (очная, очно-заочная, заочная) |
|  |  |  |

**Отчет по лабораторной работе №3**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **на тему** |  | «Переменные окружения». | | |
|  |  | (наименование темы) | | |
|  |  | Меж процессное взаимодействии | | |
| **по дисциплине** | | |  | **Системное программирование | ПМ.01 | Разработка модулей программного обеспечения для компьютерных систем** |
|  | | |  | (наименование дисциплины) |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Обучающийся** |  | Старчик Артемий Филиппович |  |  |
|  |  | (ФИО) |  | (подпись) |
| **Группа** |  | ДКИП-206Прог |  |  |
|  |  |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Преподаватель** |  | Сибирев Иван Валерьевич |  |  |
|  |  | (ФИО) |  | (подпись) |

**Москва 2025г.**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <pthread.h>

#include <unistd.h>

#define MAX\_THREADS 10

#define MAX\_FILENAME\_LEN 256

// Структура для передачи данных в поток

typedef struct {

char input\_file[MAX\_FILENAME\_LEN];

char output\_file[MAX\_FILENAME\_LEN];

int word\_count;

} ThreadData;

// Глобальные переменные

ThreadData thread\_data\_array[MAX\_THREADS];

pthread\_mutex\_t mutex = PTHREAD\_MUTEX\_INITIALIZER;

// Функция для подсчета слов в файле

void\* count\_words(void\* arg) {

ThreadData\* data = (ThreadData\*)arg;

FILE\* input = fopen(data->input\_file, "r");

if (!input) {

perror("Error opening input file");

pthread\_exit((void\*)-1);

}

FILE\* output = fopen(data->output\_file, "w");

if (!output) {

perror("Error opening output file");

fclose(input);

pthread\_exit((void\*)-1);

}

int count = 0;

char ch;

int in\_word = 0;

while ((ch = fgetc(input)) != EOF) {

if (ch == ' ' || ch == '\t') { // Исправлено: добавлен ||

in\_word = 0;

} else {

if (!in\_word) {

count++;

in\_word = 1;

}

}

}

// Защита доступа к общим данным

pthread\_mutex\_lock(&mutex);

data->word\_count = count;

pthread\_mutex\_unlock(&mutex);

fprintf(output, "Word count: %d\n", count);

fclose(input);

fclose(output);

pthread\_exit((void\*)(intptr\_t)count);

}

int main(int argc, char\* argv[]) {

if (argc < 2) {

fprintf(stderr, "Usage: %s <input\_file1> <input\_file2> ... <input\_fileN>\n", argv[0]);

return -1;

}

int num\_files = argc - 1;

if (num\_files > MAX\_THREADS) {

fprintf(stderr, "Too many files. Maximum allowed: %d\n", MAX\_THREADS);

return -1;

}

pthread\_t threads[MAX\_THREADS];

// Инициализация данных для потоков

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

snprintf(thread\_data\_array[i].input\_file, MAX\_FILENAME\_LEN, "%s", argv[i + 1]);

snprintf(thread\_data\_array[i].output\_file, MAX\_FILENAME\_LEN, "output\_%d.txt", i + 1);

thread\_data\_array[i].word\_count = 0;

// Создание потока

if (pthread\_create(&threads[i], NULL, count\_words, &thread\_data\_array[i]) != 0) {

perror("Error creating thread");

return -1;

}

}

// Ожидание завершения потоков

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

void\* status;

if (pthread\_join(threads[i], &status) != 0) {

perror("Error joining thread");

return -1;

}

printf("Thread %d finished with status: %ld\n", i + 1, (intptr\_t)status);

}

// Вывод результатов

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

printf("File: %s, Word count: %d\n", thread\_data\_array[i].input\_file, thread\_data\_array[i].word\_count);

}

// Освобождение ресурсов

pthread\_mutex\_destroy(&mutex);

return 0;

}

}



7. Контрольные вопросы

1. Функция порождения потока и особенности функции, запускаемой в порожденном потоке

* Функция порождения потока: pthread\_create().

int pthread\_create(pthread\_t \*thread, const pthread\_attr\_t \*attr, void \*(\*start\_routine)(void\*), void \*arg);

* + thread: идентификатор нового потока.
  + attr: атрибуты потока (можно передать NULL для значений по умолчанию).
  + start\_routine: функция, которую выполнит поток.
  + arg: аргумент для функции.
* Особенности функции потока:
  + Должна возвращать void\* и принимать void\* в качестве аргумента.
  + Завершается при возврате из функции или вызове pthread\_exit().
  + Ресурсы потока освобождаются только после pthread\_join() (если поток не detached).

2. Особенности POSIX API для работы с потоками

* Основные функции:
  + pthread\_create(), pthread\_join(), pthread\_detach().
  + Мьютексы (`pthreadpthread\_mutex\_\*), условные переменные (pthread\_cond\_\*).
* Особенности:
  + Потоки POSIX (NPTL) легковесны.
  + Требуется компиляция с флагом -pthread.
  + Поддержка атрибутов для настройки потоков (например, detached state, размер стека).

3. Завершение потока

* Когда происходит:
  + Функция потока возвращает значение.
  + Вызов pthread\_exit().
  + Основной поток завершается (если поток не detached).
  + Принудительная отмена через pthread\_cancel().

4. Как сделать поток отсоединенным

* Способы:
  1. При создании через атрибуты:

pthread\_attr\_t attr;

pthread\_attr\_init(&attr);

pthread\_attr\_setdetachstate(&attr, PTHREAD\_CREATE\_DETACHED);

pthread\_create(&thread, &attr, func, NULL);

* 1. После создания:

pthread\_detach(thread);

5. Особенности досрочного завершения потока

* Функция: pthread\_cancel(pthread\_t thread).
* Особенности:
  + Поток может игнорировать отмену (установка pthread\_setcancelstate()).
  + Требует обработки ресурсов через pthread\_cleanup\_push().

6. Особенности главного потока

* Главный поток — первый поток процесса.
* Особенности:
  + Если завершается до дочерних потоков, все пот
  + Рекомендуется использовать pthread\_join() для ожидания дочерних потоков.

7. Жизненный цикл потоков

1. Создание: pthread\_create().
2. Выполнение: выполнение функции start\_routine.
3. Завершение:
   * Явное: возврат из функции или pthread\_exit().
   * Принудительное: pthread\_cancel().
4. Освобождение ресурсов: после pthread\_join() или если поток detached.

8. Объект атрибутов потока

* Создание:

pthread\_attr\_t attr;

pthread\_attr\_init(&attr); // Инициализация

pthread\_attr\_setdetachstate(&attr, PTHREAD\_CREATE\_DETACHED); // Настройка

pthread\_create(&thread, &attr, func, NULL);

pthread\_attr\_destroy(&attr); // Освобождение

\*\*9. З9. Значения атрибутов потока по умолчанию

* Detached state: PTHREAD\_CREATE\_JOINABLE (требуется pthread\_join()).
* Размер стека: определяется системой.
* Планировщик: наследуется от родительского потока.

10. Функции для задания/чтения атрибутов

* Настройка:
  + pthread\_attr\_setdetachstate(), pthread\_attr\_setstacksize().
* Чтение:
  + pthread\_attr\_getdetachstate(), pthread\_attr\_getstacksize().

11. Мьютекс и его назначение

* Мьютекс — механизм синхронизации для защиты разделяемых данных.
* Назначение: обеспечивает эксклюзивный доступ к ресурсу.

12. Инициализация мьютекса

* Статическая:

pthread\_mutex\_t mutex = PTHREAD\_MUTEX\_INITIALIZER;

* Динамическая:

pthread\_mutex\_init(&mutex, NULL); // NULL = атрибуты по умолчанию

13. Функции для работы с мьютексами

* Основные функции:
  + pthread\_mutex\_lock(): блокировка.
  + pthread\_mutex\_unlock(): разблокировка.
  + pthread\_mutex\_trylock(): попытка блокировки без ожидания.
  + pthread\_mutex\_destroy(): освобождение ресурсов.

14. Условные переменные

* Назначение: координация потоков при изменении состояния данных.
* Пример: ожидание заполнения буфера.
* Используются с мьютексами для синхронизации.

16. Функции для работы с условными переменными

* pthread\_cond\_wait(): ожидание сигнала.
* pthread\_cond\_signal(): пробуждение одного потока.
* pthread\_cond\_broadcast(): пробуждение всех потоков.
* pthread\_cond\_timedwait(): ожидание с

17. Ложное пробуждение

* Причина: поток может проснуться без получения сигнала.
* Решение: использовать цикл проверки условия:

while (условие\_не\_выполнено) {

pthread\_cond\_wait(&cond, &mutex);

}

18. pthread\_cond\_broadcast и pthread\_cond\_timedwait

* pthread\_cond\_broadcast(): пробуждает все потоки, ожидающие условной переменной.
* pthread\_cond\_timedwait(): ожидает сигнал или таймаут (предотвращает бесконечное ожидание).

19. Мьютексы и условные переменные в разделяемой памяти

* Настройка атрибутов:

pthread\_mutexattr\_t mattr;

pthread\_mutexattr\_init(&mattr);

pthread\_mutexattr\_setpshared(&mattr, PTHREAD\_PROCESS\_SHARED);

pthread\_mutex\_init(&mutex, &mattr);

20. Атрибуты мьютекса

* Типы:
  + PTHREAD\_MUTEX\_NORMAL: базовый мьютекс.
  + PTHREAD\_MUTEX\_RECURSIVE: разрешает многократную блокировку.
  + PTHREAD\_MUTEX\_ERRORCHECK: проверка ошибок.
* Протоколы: PTHREAD\_PRIO\_INHERIT (наследование приоритета).