Московский Авиационный Институт

(Национальный Исследовательский Университет)

Институт №8 “Компьютерные науки и прикладная математика”

Кафедра №806 “Вычислительная математика и программирование”

Лабораторная работа №2 по курсу

«Операционные системы»

Группа: М8О-213Б-23

Студент: Иванов В. М.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 18.10.24

Москва, 2024

Постановка задачи

Вариант 18.

Составить программу на языке Си, обрабатывающую данные в многопоточном режиме. При обработкt использовать стандартные средства создания потоков операционной системы(Windows/Unix). Ограничение максимального количества потоков, работающих в один момент времени, должно быть задано ключом запуска вашей программы.

Найти образец в строке наивным алгоритмом

Общий метод и алгоритм решения

Кратко опишите системные вызовы, которые вы использовали в лабораторной работе.

Использованные системные вызовы:

- int pthread_mutex_init (pthread_mutex_t *__mutex, const pthread_mutexattr_t *__mutexattr) — создать мьютекс
- int pthread_mutex_lock() - заблокировать мьютекс
- int pthread_mutex_unlock() - разблокировать мьютекс
- int pthread_create() - создать поток
- int pthread_join() - ждать завершения потока
- void* mmap() - отобразить файл на память
- int fstat() - получить данные о файле
- int close(int fd) — закрыть файл.
- int open() - открытие/создание файла.
- int write() - вывод на экран сообщение.

Общий алгоритм:

- открыть файл и с помошью mmap «смонтировать» его в память
- инициализировать структуры данных для потоков, разбив строку на n кусочков
- ройтись по массиву созанных структур и запустить потоки
- дождаться завершения потоков
- вывести результаты и «размонтировать» файл

Код программы

main.c

```
#include <fcntl.h>

#include <unistd.h>

#include <string.h>

#include <pthread.h>

#include <sys/mman.h>

#include <sys/stat.h>


#define MAX_THREADS 10 // Максимальное количество потоков


typedef struct

{

    char *haystack;  // Основная строка
```

```
    char *needle;    // Подстрока для поиска
    long long start; // Начальный индекс для поиска
    long long end;   // Конечный индекс для поиска
    int thread_id;   // ID потока
} ThreadData;

void reverse_str(char *str)
{
    int len = strlen(str);
    for (int i = 0; i < len / 2; i++)
    {
        char tmp = str[i];
        str[i] = str[len - 1 - i];
        str[len - 1 - i] = tmp;
    }
}

void print_ll(long long num, int fd)
{
    char buf[256];
    int idx = 0;
    if (num == 0)
    {
        buf[idx++] = '0';
    }
    else
    {
        while (num)
        {
            buf[idx++] = num % 10 + '0';
            num /= 10;
        }
    }
    buf[idx] = 0;
    reverse_str(buf);
    write(fd, buf, idx);
```

```
}
int is_digit(char c)
{
    return (c >= '0' && c <= '9');
}
int str_to_i(char *str)
{
    int buf = 0;
    int len = strlen(str);
    for (int i = 0; i < len; i++)
    {
        if (!is_digit(str[i]))
            return -1;
        buf *= 10;
        buf += (str[i] - '0');
    }
    return buf;
}

pthread_mutex_t mutex;
long long found_position = -1; // Позиция найденной подстроки
int res[MAX_THREADS];

void *search_substring(void *arg)
{
    ThreadData *data = (ThreadData *)arg;

    for (long long i = data->start; i <= data->end - strlen(data->needle); i++)
    { // ищем дальше нашец границы на длинну шаблона
        if (strncmp(&data->haystack[i], data->needle, strlen(data->needle)) ==
0)
        {
            pthread_mutex_lock(&mutex);
            if (found_position == -1)
            {                       // Если еще не нашли
                found_position = i; // Записываем позицию
```

```
                res[data->thread_id] = 1;
                // printf("Подстрока найдена в потоке %d на позиции %lld\n",
data->thread_id, found_position);
            }
            pthread_mutex_unlock(&mutex);
            return NULL; // Прерываем поиск, если уже нашли
        }
    }
    res[data->thread_id] = 0;
    return NULL;
}

int main(int argc, char **argv)
{
    if (argc != 4)
    {
        write(STDERR_FILENO, "Wrong number of args!", 21);
        return -1;
    }
    int num_threads = MAX_THREADS;
    if (!(strcmp(argv[2], "-j")))
    {
        num_threads = str_to_i(argv[3]);
        if (num_threads < 0)
        {
            write(STDERR_FILENO, "Error reading number", 20);
            return -1;
        }
        if (num_threads > MAX_THREADS)
        {
            write(STDERR_FILENO, "Maximum thread count exceeded!", 30);
            return -1;
        }
    }
    else
    {
```

```
        write(STDERR_FILENO, "Unknown flag (use -j num)", 25);
        return -1;
    }
    int fd = open(argv[1], O_RDONLY);
    if (fd < 0)
    {
        write(STDERR_FILENO, "Error opening file", 18);
        return -1;
    }
    struct stat st;
    if (fstat(fd, &st))
    {
        close(fd);
        write(STDERR_FILENO, "Error in stat", 13);
        return -1;
    }
    // char* haystack = mmap(NULL,1027,PROT_READ,(MAP_PRIVATE),fd, 9999998976);
    long long len_haystack = st.st_size; // strlen не работает на 10гигабайтной
строке и падает в сегфолт :(
    char *haystack = mmap(NULL, len_haystack, PROT_READ, (MAP_PRIVATE), fd, 0);
    if (haystack == (char *)-1)
    {
        close(fd);
        write(STDERR_FILENO, "Error in MMAP", 13);
        return -1;
    }


    char *needle = "34";


    pthread_t threads[num_threads];


    ThreadData thread_data[num_threads];
    pthread_mutex_init(&mutex, NULL);


    long long chunk_size = (long long)len_haystack / num_threads;
```

```
    // Создаем потоки
    for (int i = 0; i < num_threads; i++)
    {
        thread_data[i].haystack = haystack;
        thread_data[i].needle = needle;
        thread_data[i].start = i * chunk_size;
        thread_data[i].end = (i == num_threads - 1) ? len_haystack : (i + 1) *
chunk_size + strlen(needle);
        thread_data[i].thread_id = i;

        if(pthread_create(&threads[i], NULL, search_substring, (void
*)&thread_data[i])){
            write(STDERR_FILENO, "Error creating thread!", 22);
            munmap(haystack, len_haystack);
            close(fd);
            return -1;
        };
    }

    // Ждем завершения всех потоков
    for (int i = 0; i < num_threads; i++)
    {
        pthread_join(threads[i], NULL);
    }

    if (found_position == -1)
    {
        write(1, "NOT FOUND\n", 10);
    }
    else
    {
        for (int i = 0; i < num_threads; i++)
        {
            if (res[i] == 1)
            {
                write(1, "found in thread: ", 17);
```

```
                print_ll(i, 1);
                write(1, " at position ", 13);
                print_ll(found_position, 1);
            }
        }
    }
    pthread_mutex_destroy(&mutex);
    munmap(haystack, len_haystack);
    close(fd);
    return 0;
}
```

Протокол работы программы

Тестирование:

10G строка, состоящая из "01230123...01234", находящаяся в файле input_c.txt

образец: "34"

time ./main input_c.txt -j 1

found in thread: 0 at position 10000000000

real 112,61s

user 108,12s

sys 3,30s

cpu 98%

time ./main input_c.txt -j 2

found in thread: 1 at position 10000000000

real 79,10s

user 150,56s

sys 4,49s

cpu 196%

time ./main input_c.txt -j 4 (количество логических потоков процессора)

found in thread: 3 at position 10000000000

real 72,33s

user 253,88s

sys 5,87s

cpu 359%

time ./main input_c.txt -j 6

found in thread: 5 at position 10000000000

real 74,25s

user 264,15s

sys 5,53s

cpu 363%

time ./main input_c.txt -j 8

found in thread: 7 at position 10000000000

real 68,56s

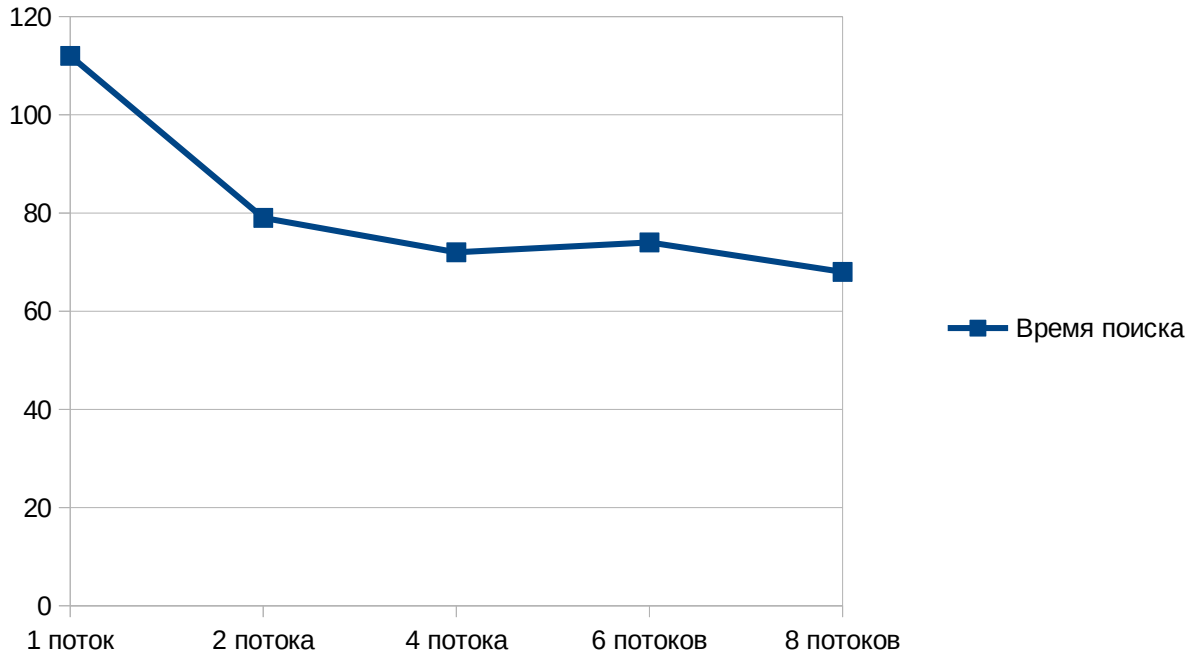
user 248,67s

sys 5,88s

cpu 371%

Сравнительная таблица:

1 поток	112 секунд
2 потока (количество физических ядер)	79 секунд
4 потока (количество логических потоков)	72 секунды
6 потоков	74 секунды
8 потоков	68 секунд



Strace:

```
    execve("./main", ["./main", "input.txt", "-j", "4"], 0x7ffcd01e7318 /* 95 vars */) = 0

    brk(NULL)                                   = 0x1004000

    arch_prctl(0x3001   /*   ARCH_???   */,   0x7ffd2bde5e90)   =   -1   EINVAL (Недопустимый аргумент)

    access("/etc/ld.so.preload", R_OK)       = -1 ENOENT (Нет такого файла или каталога)

    openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3

    fstat(3, {st_mode=S_IFREG|0644, st_size=99383, ...}) = 0

    mmap(NULL, 99383, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f962e916000

    close(3)                                    = 0

    openat(AT_FDCWD, "/lib64/libc.so.6", O_RDONLY|O_CLOEXEC) = 3

    read(3,                   "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0P\227\2\0\0\0\0\0"..., 832) = 832

    pread64(3,                "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0\0"..., 784, 64) = 784

    pread64(3,                    "\4\0\0\0                   \0\0\0\5\0\0\0GNU\0\2\0\0\300\4\0\0\0\3\0\0\0\0\0\0\0"..., 48, 848) = 48

    pread64(3,                                  "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0T\247\253\1\356\366\342\334\242\306\260\332\270\306V\241"..., 68, 896) = 68

    fstat(3, {st_mode=S_IFREG|0755, st_size=2592552, ...}) = 0
```

```
    mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7f962e914000

    pread64(3,                "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\
0\0\0\0\0\0\0"..., 784, 64) = 784

    mmap(NULL,   2133936,   PROT_READ,   MAP_PRIVATE|MAP_DENYWRITE,   3,   0)   =
0x7f962e600000

    mprotect(0x7f962e628000, 1892352, PROT_NONE) = 0

    mmap(0x7f962e628000,  1527808,  PROT_READ|PROT_EXEC,  MAP_PRIVATE|MAP_FIXED|
MAP_DENYWRITE, 3, 0x28000) = 0x7f962e628000

    mmap(0x7f962e79d000,     360448,     PROT_READ,     MAP_PRIVATE|MAP_FIXED|
MAP_DENYWRITE, 3, 0x19d000) = 0x7f962e79d000

    mmap(0x7f962e7f6000,  24576,  PROT_READ|PROT_WRITE,  MAP_PRIVATE|MAP_FIXED|
MAP_DENYWRITE, 3, 0x1f5000) = 0x7f962e7f6000

    mmap(0x7f962e7fc000,  53168,  PROT_READ|PROT_WRITE,  MAP_PRIVATE|MAP_FIXED|
MAP_ANONYMOUS, -1, 0) = 0x7f962e7fc000

    close(3)                                = 0

    mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7f962e912000

    arch_prctl(ARCH_SET_FS, 0x7f962e915600) = 0

    set_tid_address(0x7f962e9158d0)         = 21693

    set_robust_list(0x7f962e9158e0, 24)     = 0

    rseq(0x7f962e915fa0, 0x20, 0, 0x53053053) = 0

    mprotect(0x7f962e7f6000, 16384, PROT_READ) = 0

    mprotect(0x403000, 4096, PROT_READ)     = 0

    mprotect(0x7f962e963000, 8192, PROT_READ) = 0

    prlimit64(0,         RLIMIT_STACK,         NULL,         {rlim_cur=8192*1024,
rlim_max=RLIM64_INFINITY}) = 0

    munmap(0x7f962e916000, 99383)           = 0

    openat(AT_FDCWD, "input.txt", O_RDONLY) = 3

    fstat(3, {st_mode=S_IFREG|0644, st_size=10000002, ...}) = 0

    mmap(NULL, 10000002, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f962dc00000

    rt_sigaction(SIGRT_1,        {sa_handler=0x7f962e686f70,        sa_mask=[],
sa_flags=SA_RESTORER|SA_ONSTACK|SA_RESTART|SA_SIGINFO,
sa_restorer=0x7f962e63e6f0}, NULL, 8) = 0

    rt_sigprocmask(SIG_UNBLOCK, [RTMIN RT_1], NULL, 8) = 0

    mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0)
= 0x7f962d3ff000

    mprotect(0x7f962d400000, 8388608, PROT_READ|PROT_WRITE) = 0

    getrandom("\xdf\x8d\x4b\xe7\xd6\x6a\xe6\xd7", 8, GRND_NONBLOCK) = 8

    brk(NULL)                               = 0x1004000

    brk(0x1025000)                          = 0x1025000
```

```
     rt_sigprocmask(SIG_BLOCK, ~[], [], 8)   = 0

     clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|
CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID,
child_tid=0x7f962dbff910,       parent_tid=0x7f962dbff910,       exit_signal=0,
stack=0x7f962d3ff000,      stack_size=0x7fff00,      tls=0x7f962dbff640}      =>
{parent_tid=[21694]}, 88) = 21694

     rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0

     mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0)
= 0x7f962cbfe000

     mprotect(0x7f962cbff000, 8388608, PROT_READ|PROT_WRITE) = 0

     rt_sigprocmask(SIG_BLOCK, ~[], [], 8)   = 0

     clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|
CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID,
child_tid=0x7f962d3fe910,       parent_tid=0x7f962d3fe910,       exit_signal=0,
stack=0x7f962cbfe000,      stack_size=0x7fff00,      tls=0x7f962d3fe640}      =>
{parent_tid=[21695]}, 88) = 21695

     rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0

     mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0)
= 0x7f962c3fd000

     mprotect(0x7f962c3fe000, 8388608, PROT_READ|PROT_WRITE) = 0

     rt_sigprocmask(SIG_BLOCK, ~[], [], 8)   = 0

     clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|
CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID,
child_tid=0x7f962cbfd910,       parent_tid=0x7f962cbfd910,       exit_signal=0,
stack=0x7f962c3fd000,      stack_size=0x7fff00,      tls=0x7f962cbfd640}      =>
{parent_tid=[21696]}, 88) = 21696

     rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0

     mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0)
= 0x7f962bbfc000

     mprotect(0x7f962bbfd000, 8388608, PROT_READ|PROT_WRITE) = 0

     rt_sigprocmask(SIG_BLOCK, ~[], [], 8)   = 0

     clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|
CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID,
child_tid=0x7f962c3fc910,       parent_tid=0x7f962c3fc910,       exit_signal=0,
stack=0x7f962bbfc000,      stack_size=0x7fff00,      tls=0x7f962c3fc640}      =>
{parent_tid=[21697]}, 88) = 21697

     rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0

     futex(0x7f962dbff910, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 21694, NULL,
FUTEX_BITSET_MATCH_ANY) = 0

     write(1, "found in thread: ", 17)       = 17

     write(1, "3", 1)                        = 1

     write(1, " at position ", 13)           = 13

     write(1, "9999999", 7)                  = 7

     munmap(0x7f962dc00000, 10000002)        = 0

     exit_group(0)                           = ?
```

```
+++ exited with 0 +++
```

Вывод

Я научился созадвать потоки и многопоточные приложения в Linux, оценивать их применимость и производительность, Так же я научился отображать файл на память без его непосредственной загрузки с помошью mmap и узнал, чт strlen крашит программу при длинне строки больше чем MAX_UNSIGNED_INT