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: Савинов Н. О.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 28.12.24

Москва, 2024

Постановка задачи

Вариант 1.

Отсортировать массив целых чисел при помощи битонической сортировки.

Общий метод и алгоритм решения

Использованные системные вызовы:

● int int_to_str(int num, char *buffer, int buffer_size) – преобразование числа в строку и записи в буфер

● void bitonic_merge(int arr[], int low, int count, int direction) - выполнение битонического слияния

● void bitonic_sort(int arr[], int low, int count, int direction, int *active_processes, int max_processes) – выполнение битонической сортировки

● int is_power_of_two(int n) – проверка на то, является ли число степенью 2

● void print_array(int arr[], int size) – печать массива

Программа выполняет битоническую сортировку с параллельной обработкой, используя процессы для сортировки массива чисел. Программа принимает два аргумента командной строки: размер массива чисел (n) и максимальное количество одновременно работающих процессов (max_processes). Программа проверяет, что размер массива является степенью двойки (необходимо для корректной работы битонической сортировки). Если размер массива не является степенью двойки, программа завершает выполнение с ошибкой. Программа создает массив целых чисел arr, заполняет его случайными значениями с помощью rand(), а затем выводит начальный массив на экран. Алгоритм сортировки использует битоническую сортировку, которая делит массив на две половины, и затем выполняет слияние этих половин с определенным направлением (возрастающий или убывающий порядок). Битоническая сортировка выполняется рекурсивно, и для каждого шага сортировки запускаются новые процессы для параллельного выполнения слияния.

- **Битоническое слияние** (bitonic_merge): Функция, которая выполняет слияние двух отсортированных частей массива в один отсортированный массив. Направление слияния зависит от параметра direction.
- **Битоническая сортировка** (bitonic_sort): Рекурсивная функция, которая делит массив на две половины и сортирует их параллельно в два потока (процесса), если количество активных процессов меньше заданного предела max_processes. В противном случае она продолжает сортировку в текущем процессе. Каждый процесс выполняет часть работы: одна половина массива сортируется в порядке возрастания, другая — в порядке убывания.
- **Параллельность**: Процессы создаются с помощью fork(). Каждый дочерний процесс выполняет свою часть сортировки и вызывает bitonic_sort для своей половины массива. После выполнения дочернего процесса главный процесс ждет его завершения с помощью waitpid().

После выполнения сортировки программа выводит отсортированный массив на экран и освобождает память, используя munmap().

Код программы

main.c

```
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <fcntl.h>
#include <errno.h>
#include <string.h>
#include <time.h>
#include <stdint.h>
#include <sys/mman.h>

#define MAX_PROCESSES 8 // Максимальное количество процессов

// Функция для обмена элементов
void swap(int *a, int *b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}

// Функция для преобразования числа в строку и записи в буфер
int int_to_str(int num, char *buffer, int buffer_size) {
    int len = 0;
    int temp = num;
    do {
        if (len >= buffer_size - 1) break;
        buffer[len++] = '0' + (temp % 10);
        temp /= 10;
    } while (temp > 0);
```

```
    // Реверс строки
    for (int i = 0; i < len / 2; i++) {
        char tmp = buffer[i];
        buffer[i] = buffer[len - i - 1];
        buffer[len - i - 1] = tmp;
    }
    buffer[len] = '\0'; // Добавляем нулевой символ
    return len;
}

// Функция для выполнения битонического слияния
void bitonic_merge(int arr[], int low, int count, int direction) {
    if (count > 1) {
        int k = count / 2;
        for (int i = low; i < low + k; i++) {
            if (direction == (arr[i] > arr[i + k])) {
                swap(&arr[i], &arr[i + k]);
            }
        }
        bitonic_merge(arr, low, k, direction);
        bitonic_merge(arr, low + k, k, direction);
    }
}

// Функция для выполнения битонической сортировки
void bitonic_sort(int arr[], int low, int count, int direction, int *active_processes, int max_processes) {
    if (count > 1) {
        int k = count / 2;
        pid_t pid1 = -1, pid2 = -1;
```

```
int process_created = 0;


if (*active_processes < max_processes) {
    pid1 = fork();
    if (pid1 < 0) {
        write(2, "Error: fork failed\n", 19);
        _exit(1);
    }
    if (pid1 == 0) { // Дочерний процесс
        (*active_processes)++;
        bitonic_sort(arr, low, k, 1, active_processes, max_processes);
        _exit(0);
    }
    process_created = 1;
} else {
    bitonic_sort(arr, low, k, 1, active_processes, max_processes);
}


if (*active_processes < max_processes) {
    pid2 = fork();
    if (pid2 < 0) {
        write(2, "Error: fork failed\n", 19);
        _exit(1);
    }
    if (pid2 == 0) { // Дочерний процесс
        (*active_processes)++;
        bitonic_sort(arr, low + k, k, 0, active_processes, max_processes);
        _exit(0);
    }
    process_created = 1;
```

```
        } else {
            bitonic_sort(arr, low + k, k, 0, active_processes, max_processes);
        }

        // Ожидание завершения дочерних процессов
        if (process_created) {
            int status;
            if (pid1 > 0) {
                if (waitpid(pid1, &status, 0) == -1) {
                    write(2, "Error: waitpid failed\n", 23);
                }
                (*active_processes)--;
            }
            if (pid2 > 0) {
                if (waitpid(pid2, &status, 0) == -1) {
                    write(2, "Error: waitpid failed\n", 23);
                }
                (*active_processes)--;
            }
        }
        bitonic_merge(arr, low, count, direction);
    }
}

// Функция для проверки, является ли число степенью 2
int is_power_of_two(int n) {
    return (n > 0) && ((n & (n - 1)) == 0);
}

// Функция для печати массива
```

```
void print_array(int arr[], int size) {
    char buffer[128];
    for (int i = 0; i < size; i++) {
        int len = int_to_str(arr[i], buffer, sizeof(buffer));
        if (write(1, buffer, len) == -1) {
            write(2, "Error: write failed\n", 20);
        }
        if (write(1, " ", 1) == -1) {
            write(2, "Error: write failed\n", 20);
        }
    }
    write(1, "\n", 1);
}

int main(int argc, char *argv[]) {
    if (argc < 3) {
        const char *usage = "Usage: ./bitonic_sort <array size> <max processes>\n";
        write(1, usage, strlen(usage));
        return 1;
    }

    int n = atoi(argv[1]);
    int max_processes = atoi(argv[2]);

    if (!is_power_of_two(n)) {
        const char *error_msg = "Error: Array size must be a power of 2\n";
        write(2, error_msg, strlen(error_msg));
        return 1;
    }
```

```
    if (max_processes <= 0) {
        max_processes = MAX_PROCESSES;
    }

    int *arr = mmap(NULL, n * sizeof(int), PROT_READ | PROT_WRITE, MAP_SHARED |
MAP_ANONYMOUS, -1, 0);
    if (arr == MAP_FAILED) {
        write(2, "Error: mmap failed\n", 18);
        return 1;
    }

    srand(time(NULL));
    for (int i = 0; i < n; i++) {
        arr[i] = rand() % 100;
    }

    write(1, "Initial array:\n", 15);
    print_array(arr, n);

    int active_processes = 1;
    bitonic_sort(arr, 0, n, 1, &active_processes, max_processes);

    write(1, "\nSorted array:\n", 15);
    print_array(arr, n);

    if (munmap(arr, n * sizeof(int)) == -1) {
        write(2, "Error: munmap failed\n", 20);
    }

    return 0;
```

```
}
```

Протокол работы программы

Тестирование:

```
Starting program: /home/artemdelgray/Загрузки/Telegram Desktop/Лаба2/l2 4 2
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
Initial array:
9 52 24 28
[Detaching after fork from child process 4549]
[Detaching after fork from child process 4550]

Sorted array:
9 24 28 52
[Inferior 1 (process 4548) exited normally]
```

Strace:

```
strace -f ./l2 4 2

execve("./l2", ["./l2", "4", "2"], 0x7ffc173a1e18 /* 46 vars */) = 0

brk(NULL)                               = 0x61cda8998000

arch_prctl(0x3001 /* ARCH_??? */, 0x7ffcfbcff630) = -1 EINVAL (Недопустимый аргумент)

mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x784b990bf000

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

newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=58047, ...}, AT_EMPTY_PATH) = 0

mmap(NULL, 58047, PROT_READ, MAP_PRIVATE, 3, 0) = 0x784b990b0000

close(3)                                = 0

openat(AT_FDCWD, "/lib/x86_64-linux-gnu/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\237\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"..., 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\0I\17\357\204\3$\f\221\2039x\324\224\323\236S"...,
68, 896) = 68

newfstatat(3, "", {st_mode=S_IFREG|0755, st_size=2220400, ...}, AT_EMPTY_PATH) = 0

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, 2264656, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x784b98e00000

mprotect(0x784b98e28000, 2023424, PROT_NONE) = 0

mmap(0x784b98e28000, 1658880, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0x28000) = 0x784b98e28000

mmap(0x784b98fbd000, 360448, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1bd000) =
0x784b98fbd000

mmap(0x784b99016000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0x215000) = 0x784b99016000

mmap(0x784b9901c000, 52816, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1,
0) = 0x784b9901c000

close(3)                                = 0

mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x784b990ad000

arch_prctl(ARCH_SET_FS, 0x784b990ad740) = 0

set_tid_address(0x784b990ada10)         = 4563
```

```
set_robust_list(0x784b990ada20, 24)     = 0

rseq(0x784b990ae0e0, 0x20, 0, 0x53053053) = 0

mprotect(0x784b99016000, 16384, PROT_READ) = 0

mprotect(0x61cda7b24000, 4096, PROT_READ) = 0

mprotect(0x784b990f9000, 8192, PROT_READ) = 0

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

munmap(0x784b990b0000, 58047)           = 0

mmap(NULL, 16, PROT_READ|PROT_WRITE, MAP_SHARED|MAP_ANONYMOUS, -1, 0) = 0x784b990f8000

write(1, "Initial array:\n", 15Initial array:

)        = 15

write(1, "8", 18)                        = 1

write(1, " ", 1 )                        = 1

write(1, "80", 280)                       = 2

write(1, " ", 1 )                        = 1

write(1, "32", 232)                       = 2

write(1, " ", 1 )                        = 1

write(1, "69", 269)                       = 2

write(1, " ", 1 )                        = 1

write(1, "\n", 1

)                       = 1

clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLDstrace: Process
4564 attached

, child_tidptr=0x784b990ada10) = 4564

[pid  4563] clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD
<unfinished ...>

[pid  4564] set_robust_list(0x784b990ada20, 24) = 0

[pid  4564] exit_group(0)               = ?

[pid  4564] +++ exited with 0 +++

strace: Process 4565 attached

[pid  4565] set_robust_list(0x784b990ada20, 24) = 0

[pid  4565] exit_group(0)               = ?

[pid  4565] +++ exited with 0 +++

<... clone resumed>, child_tidptr=0x784b990ada10) = 4565
```

```
--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=4564, si_uid=1000, si_status=0,
si_utime=0, si_stime=0} ---

wait4(4564, [{WIFEXITED(s) && WEXITSTATUS(s) == 0}], 0, NULL) = 4564

wait4(4565, [{WIFEXITED(s) && WEXITSTATUS(s) == 0}], 0, NULL) = 4565

write(1, "\nSorted array:\n", 15

Sorted array:

)       = 15

write(1, "8", 18)                        = 1

write(1, " ", 1 )                        = 1

write(1, "32", 232)                       = 2

write(1, " ", 1 )                        = 1

write(1, "69", 269)                       = 2

write(1, " ", 1 )                        = 1

write(1, "80", 280)                       = 2

write(1, " ", 1 )                        = 1

write(1, "\n", 1

)                       = 1

munmap(0x784b990f8000, 16)              = 0

exit_group(0)                           = ?

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

Вывод

В этой лабораторной работе реализована сортировка массива путём использования метода битонической сортировки. Программа демонстрирует мощь параллельной обработки, особенно на больших массивах, где многопроцессорная сортировка может значительно сократить время выполнения. Однако она требует осторожного подхода к выбору размера массива и количества процессов для эффективного использования системных ресурсов.