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-210Б-23

Студент: Жаворонков Н. Д.

Преподаватель: Бахарев В. Д.

Оценка: _____

Дата: 27.12.24

Москва
2024

Постановка задачи

Вариант 10.

Данное задание направлено на освоение фундаментальных аспектов многопоточного программирования, а именно:

- **Мастерство управления потоками:** Приобретение практических навыков по созданию, запуску и контролю потоков в операционной системе.
- **Искусство синхронизации:** Развитие умения обеспечивать корректную и безопасную синхронизацию между параллельно выполняемыми потоками.

Задание:
Разработать на языке Си многопоточное приложение для обработки данных, опираясь на стандартные механизмы создания потоков, предоставляемые операционными системой Unix. Программа должна обладать следующими характеристиками:

1. **Гибкость:** Максимальное количество потоков, которые могут выполняться одновременно, должно задаваться при запуске программы через командную строку.
2. **Прозрачность:** Программа должна уметь демонстрировать количество используемых потоков с помощью стандартных средств операционной системы, обеспечивая наблюдаемость ее работы.
3. **Аналитичность:** Необходимо провести исследование влияния входных данных и числа потоков на ускорение и эффективность алгоритма, результаты которого должны быть представлены в отчете с подробными объяснениями.

Общий метод и алгоритм решения

1. **Чтение файла:** Родительский процесс считывает числа из файла построчно.
2. **Создание дочернего процесса:** Для каждого прочитанного числа родительский процесс создает дочерний процесс с помощью системного вызова fork().
3. **Проверка на простоту:** Дочерний процесс проверяет число на простоту. Алгоритм проверки:
    - Если число меньше 2, то оно не является простым.
    - Если число равно 2, то оно является простым.
    - Если число больше 2, то необходимо проверить, делится ли оно на числа от 2 до квадратного корня из числа. Если делится, то число составное.
4. **Вывод результата:**
    - Если число составное, то дочерний процесс выводит его в стандартный поток вывода. o Если число отрицательное или простое, то дочерний процесс завершается.
5. **Завершение родительского процесса:** Родительский процесс ожидает завершения всех дочерних процессов с помощью системного вызова wait(). После завершения всех дочерних процессов родительский процесс также завершается.

Использованные системные вызовы:

- int pthread_create(pthread_t *thread, const pthread_attr_t *attr, void*(*start_routine) (void*), void *arg)
- int pthread_join(pthread_t threads, void ** value)
- int pthread_mutex_init(pthread_mutex_t *mutex)
- int pthread_mutex_lock(pthread_mutex_t *mutex)
- int pthread_mutex_unlock(pthread_mutex_t *mutex)
- int pthread_mutex_destroy(pthread_mutex_t *mutex)

Программа реализует сортировку Бэтчера. Единственным аргументом программы является максимальное количество одновременно работающих потоков. Во время исполнения генерируется массив, заполненный случайными элементами.

Число потоков	Время выполнения (мс)	Ускорение	Эффективность
1	631	1.00	1.00
2	295	0.46	0.31
6	170	0.26	0.05
8	132	0.21	0.03

Код программы

main.c

```
/* Supertos Industries ( 2012 - 2024 )
 * Author: Supertos, 2024
 *
 * MAI Operating Systems course. Exercise 2.
 */
#include <stdio.h>
#include <stddef.h>
#include <stdlib.h>
#include <pthread.h>
#include <math.h>

#include <sys/time.h>


typedef struct ArrayPassInfo ArrayPassInfo;
struct ArrayPassInfo {
size_t mask;
size_t offset;
size_t dist;
size_t start;
size_t end;
size_t arrSize;
int* arr;
};

void swapInt( int* a, int* b ) {
int temp = *b;
*b = *a;
*a = temp;
}

/* One array pass */
void* sortArrayPassFraction( void* infoP ) {
if( !infoP ) return NULL;

ArrayPassInfo* info = (ArrayPassInfo*)infoP;

size_t mask = info->mask;
size_t offset = info->offset;
size_t dist = info->dist;
size_t arrSize = info->arrSize;
size_t start = info->start;
size_t end = info->end;
int* arr = info->arr;


for( size_t i = start; i < end; ++i ) {
```

```
        if( ( i & mask ) != offset ) continue; // Skip elements that are in even or odd
blocks, to avoid overlapping.
        if( i + dist < arrSize && arr[i] > arr[i + dist] ) swapInt( &arr[i], &arr[i +
dist] );
}

return NULL;
}

double sortArrayPass( size_t mask, size_t offset, size_t dist, size_t threads, int arr[],
size_t arrSize, size_t realArrSize, ArrayPassInfo* infoTable, pthread_t* threadlist )
{

   size_t elementsLeft = arrSize - dist;
   size_t start = 0;

for( size_t threadNo = threads; threadNo > 0; --threadNo ) {
        size_t elements = elementsLeft / threadNo;

        elementsLeft -= elements;

        infoTable[threadNo - 1] = (ArrayPassInfo){
                .mask = mask,
                .offset = offset,
                .dist = dist,

                .start = start,
                .end = start + elements,
                .arrSize = realArrSize,
                .arr = arr
        };

        pthread_create( &threadlist[threadNo - 1], NULL, sortArrayPassFraction,
&infoTable[threadNo - 1] );
        start += elements;
}

struct timeval begin;
gettimeofday( &begin, NULL );
for( size_t threadNo = threads; threadNo > 0; --threadNo ) {
        pthread_join( threadlist[threadNo - 1], NULL );
}
struct timeval end;
gettimeofday( &end, NULL );

double time_taken;

   time_taken = (end.tv_sec - begin.tv_sec);
   time_taken += (end.tv_usec - begin.tv_usec) * 1e-6;
return time_taken;
}

double batcherSort( int arr[], size_t size, size_t threads, ArrayPassInfo* infoTable,
pthread_t* threadlist ) {
size_t batcherSize = pow( 2, ceil(log2( size )));

double procTime = 0;

for (int mask = batcherSize; mask > 0; mask /= 2) {
        procTime += sortArrayPass( mask, 0, mask, threads, arr, batcherSize, size,
infoTable, threadlist );
        procTime += sortArrayPass( mask, mask, batcherSize - mask, threads, arr,
batcherSize, size, infoTable, threadlist );
```

```
    }

return procTime;
}

int* generateRandomArray(int size) {
    int* arr = (int*)malloc(sizeof(int) * size);
    if (arr == NULL) {
        return NULL;
    }
    for (int i = 0; i < size; i++) {
        arr[i] = rand() / ( RAND_MAX / 2 );
    }

    return arr;
}
int* generateBitonicSequence(int size) {
    if (size <= 0) return NULL;
    int* arr = (int*)malloc(sizeof(int) * size);
    if (arr == NULL) return NULL;

    int mid = size / 2;
    int current = rand() % 100;

    for(int i = 0; i < mid; ++i) {
        arr[i] = current;
        current += rand() % 3;
    }

    for(int i = mid; i < size; ++i) {
        arr[i] = current;
        current -= rand() % 3;
        if(current < 0)
        {
            current = 0;
        }
    }
    return arr;
}

int main( size_t argc, char** argv ) {
    srand(time(NULL));

size_t tot = 50;

size_t threads;
sscanf( argv[1], "%zu", &threads );
printf( "Threads: %zu...", threads );

int* arr = generateBitonicSequence(32768 * 8);
printf( "Array Generated!\n" );


ArrayPassInfo* infoTable = (ArrayPassInfo*)malloc( sizeof( ArrayPassInfo ) *
threads );
pthread_t* threadlist = (pthread_t*)malloc( sizeof( pthread_t ) * threads );


double procTime = 0;

for( size_t i = 0; i < tot; ++i ) {
            double ms = batcherSort( arr, 32768 * 8, threads, infoTable, threadlist );
```

```
        procTime += ms;
}
free( arr );
free( infoTable );


printf( "Tot time: %lf s\n", procTime );
printf( "Avg time: %lf s\n", procTime / (double)tot );

return 0;
}
```

Протокол работы программы

Тестирование:

```
supertos@DESKTOP-77RBNCB:~/oslab$ ./main.out 1
```

Strace :

```
execve("./a.out", ["./a.out", "1"], 0x7ffd86b6d7e8 /* 23 vars */) = 0
brk(NULL)                               = 0x562611f67000
arch_prctl(0x3001 /* ARCH_??? */, 0x7ffe18702a10) = -1 EINVAL (Invalid argument)
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f222f850000
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=17571, ...}, AT_EMPTY_PATH) = 0
mmap(NULL, 17571, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f222f84b000
close(3)                                = 0
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libm.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\0\0\0\0\0\0\0\0"..., 832) = 832
newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=940560, ...}, AT_EMPTY_PATH) = 0
mmap(NULL, 942344, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f222f764000
mmap(0x7f222f772000, 507904, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0xe000) = 0x7f222f772000
mmap(0x7f222f7ee000, 372736, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x8a000) =
0x7f222f7ee000
mmap(0x7f222f849000, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0xe4000) = 0x7f222f849000
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"..., 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"..., 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"..., 784, 64) = 784
mmap(NULL, 2264656, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f222f53b000
mprotect(0x7f222f563000, 2023424, PROT_NONE) = 0
mmap(0x7f222f563000, 1658880, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,
3, 0x28000) = 0x7f222f563000
mmap(0x7f222f6f8000, 360448, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1bd000) =
0x7f222f6f8000
mmap(0x7f222f751000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0x215000) = 0x7f222f751000
mmap(0x7f222f757000, 52816, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -
1, 0) = 0x7f222f757000
close(3)                                = 0
mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f222f538000
arch_prctl(ARCH_SET_FS, 0x7f222f538740) = 0
set_tid_address(0x7f222f538a10)         = 7576
set_robust_list(0x7f222f538a20, 24)     = 0
```

```
rseq(0x7f222f5390e0, 0x20, 0, 0x53053053) = 0
mprotect(0x7f222f751000, 16384, PROT_READ) = 0
mprotect(0x7f222f849000, 4096, PROT_READ) = 0
mprotect(0x5625e5f36000, 4096, PROT_READ) = 0
mprotect(0x7f222f88a000, 8192, PROT_READ) = 0
prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0
munmap(0x7f222f84b000, 17571)           = 0
newfstatat(1, "", {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0), ...}, AT_EMPTY_PATH) = 0
getrandom("\x45\x4d\x71\xee\x3d\x22\xed\x43", 8, GRND_NONBLOCK) = 8
brk(NULL)                               = 0x562611f67000
brk(0x562611f88000)                     = 0x562611f88000
mmap(NULL, 1052672, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7f222f437000
write(1, "Threads: 1...Array Generated!\n", 30Threads: 1...Array Generated!
) = 30
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 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=0x7f222f436910,
parent_tid=0x7f222f436910, exit_signal=0, stack=0x7f222ec36000, stack_size=0x7fff00, tls=0x7f222f436640} =>
{parent_tid=[0]}, 88) = 7582
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 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=0x7f222f436910,
parent_tid=0x7f222f436910, exit_signal=0, stack=0x7f222ec36000, stack_size=0x7fff00, tls=0x7f222f436640} =>
{parent_tid=[0]}, 88) = 7583
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 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=0x7f222f436910,
parent_tid=0x7f222f436910, exit_signal=0, stack=0x7f222ec36000, stack_size=0x7fff00, tls=0x7f222f436640} =>
{parent_tid=[0]}, 88) = 7584
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 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=0x7f222f436910,
parent_tid=0x7f222f436910, exit_signal=0, stack=0x7f222ec36000, stack_size=0x7fff00, tls=0x7f222f436640} =>
{parent_tid=[0]}, 88) = 7585
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 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=0x7f222f436910,
parent_tid=0x7f222f436910, exit_signal=0, stack=0x7f222ec36000, stack_size=0x7fff00, tls=0x7f222f436640} =>
{parent_tid=[7586]}, 88) = 7586
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 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=0x7f222f436910,
parent_tid=0x7f222f436910, exit_signal=0, stack=0x7f222ec36000, stack_size=0x7fff00, tls=0x7f222f436640} =>
{parent_tid=[0]}, 88) = 7587
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 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=0x7f222f436910,
parent_tid=0x7f222f436910, exit_signal=0, stack=0x7f222ec36000, stack_size=0x7fff00, tls=0x7f222f436640} =>
{parent_tid=[0]}, 88) = 7588
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 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=0x7f222f436910,
```

```
parent_tid=0x7f222f436910, exit_signal=0, stack=0x7f222ec36000, stack_size=0x7fff00, tls=0x7f222f436640} =>
{parent_tid=[0]}, 88) = 7589
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 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=0x7f222f436910,
parent_tid=0x7f222f436910, exit_signal=0, stack=0x7f222ec36000, stack_size=0x7fff00, tls=0x7f222f436640} =>
{parent_tid=[0]}, 88) = 7590
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 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=0x7f222f436910,
parent_tid=0x7f222f436910, exit_signal=0, stack=0x7f222ec36000, stack_size=0x7fff00, tls=0x7f222f436640} =>
{parent_tid=[0]}, 88) = 7591
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 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=0x7f222f436910,
parent_tid=0x7f222f436910, exit_signal=0, stack=0x7f222ec36000, stack_size=0x7fff00, tls=0x7f222f436640} =>
{parent_tid=[0]}, 88) = 7592
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
munmap(0x7f222f437000, 1052672)         = 0
write(1, "Tot time: 0.000010 s\n", 21Tot time: 0.000010 s
)  = 21
write(1, "Avg time: 0.000010 s\n", 21Avg time: 0.000010 s
)  = 21
exit_group(0)                           = ?
+++ exited with 0 +++
```

Вывод

В ходе написания данной лабораторной работы я научился создавать программы, работающие с несколькими потоками, а также синхронизировать их между собой. В результате тестирования программы, я проанализировал каким образом количество потоков влияет на эффективность и ускорение работы программы. Оказалось, что большое количество потоков даёт хорошее ускорение на больших количествах входных данных, но эффективность использования ресурсов находится на приемлемом уровне только на небольшом количестве потоков, не превышающем количества логических ядер процессора. Лабораторная работа была довольно интересна, так как я впервые работал с многопоточностью и синхронизацией на СИ.