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-215Б-23

Студент: Дехтеренко Д.С.

Преподаватель: Миронов Е.С.

Оценка: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата: 28.11.24

Москва, 2024

**Постановка задачи**

**Вариант 15.**

Составить программу на языке Си, обрабатывающую данные в многопоточном режиме. При обработке 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.

Так же необходимо уметь продемонстрировать количество потоков, используемое вашей программой с помощью стандартных средств операционной системы.

В отчете привести исследование зависимости ускорения и эффективности алгоритма от входных данных и количества потоков. Получившиеся результаты необходимо объяснить.

Есть колода из 52 карт, рассчитать экспериментально (метод Монте-Карло) вероятность того, что сверху лежат две одинаковых карты. Количество раундов задаётся ключом программы

**Общий метод и алгоритм решения**

Использованные системные вызовы:

**pthread\_create(pthread\_t \*thread, const pthread\_attr\_t \*attr, void \*(\*start\_routine) (void \*), void \*arg);** – создание нового потока.

**pthread\_join(pthread\_t thread, void \*\*retval);** - ожидание завершения указанного потока.

Напишем функцию do\_experiment, которую будут выполнять некоторое количество потоков. В ней будем случайным образом генерировать колоду карт в массиве, и проверять два последних карты в массиве на равенство номинала определённое количество раундом. Раунды равномерное распределим по потокам. Количество раундов и количество совпадений будем хранить в массиве, динамически выделенном в основном потоке. Ссылки на его ячейки будем передавать при создании потоков. Взаимоисключения не понадобятся, так как все ячейки разные. При обработке завершения потоков будем считывать итоговые количества совпадений и на их основе высчитывать вероятность.

Важно не забывать обрабатывать ошибки и освобождать динамически выделенную память.

**Код программы**

#include <stdlib.h>

#include <stdio.h>

#include <pthread.h>

#include <sys/time.h>

#include <unistd.h>

#define DECK\_SIZE 52

void\* do\_experiment(void\* arg) {

int deck[DECK\_SIZE];

int success\_count = 0;

int\* rounds\_and\_res = (int\*)arg;

int round\_thread\_count = rounds\_and\_res[0];

unsigned int seed = time(NULL) ^ pthread\_self();

for (int i = 0; i < round\_thread\_count; i++) {

for (int j = 0; j < DECK\_SIZE; j++)

deck[j] = rand\_r(&seed) % 14;

if (deck[DECK\_SIZE - 2] == deck[DECK\_SIZE - 1])

success\_count++;

}

rounds\_and\_res[1] = success\_count;

return NULL;

}

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

struct timeval start, end;

gettimeofday(&start, NULL);

if (argc != 3) {

perror("Usage: <thread\_count> <max\_rounds>");

return 1;

}

int thread\_count = atoi(argv[1]) - 1;

if (thread\_count < 2)

{

perror("It's necessary to have 2 and more threads");

return 2;

}

int max\_rounds = atoi(argv[2]);

int base = max\_rounds / thread\_count;

int dop = max\_rounds % thread\_count;

int\* rounds\_and\_res = (int\*)malloc(sizeof(int) \* thread\_count \* 2);

if (!rounds\_and\_res)

{

perror("Memory allocation failed");

return 3;

}

pthread\_t\* th = (pthread\_t\*)malloc(thread\_count \* sizeof(pthread\_t));

if(!th)

{

free(rounds\_and\_res);

perror("Memory allocation failed");

return 3;

}

double probability = 0.0;

for (int i = 0; i < thread\_count; i++)

{

rounds\_and\_res[2 \* i] = base + ((i < dop) ? 1 : 0);

if (pthread\_create(&th[i], NULL, do\_experiment, &rounds\_and\_res[2 \* i]) != 0)

{

perror("Failed to create thread");

free(rounds\_and\_res);

free(th);

return 4;

}

}

for (int i = 0; i < thread\_count; i++)

{

if (pthread\_join(th[i], NULL) != 0)

{

perror("Failed to join thread");

free(rounds\_and\_res);

free(th);

return 5;

}

probability += rounds\_and\_res[2 \* i + 1];

}

free(rounds\_and\_res);

free(th);

probability = probability / max\_rounds \* 100.0;

printf("Probability = %.2f%%\n", probability);

gettimeofday(&end, NULL);

long seconds = end.tv\_sec - start.tv\_sec;

long microseconds = end.tv\_usec - start.tv\_usec;

int milliseconds = (seconds \* 1000) + (microseconds / 1000.0);

printf("Program executed in %d ms.\n", milliseconds);

return 0;

}

**Протокол работы программы**

**Тест 1:**

luckyabatur@Luckyabatur:~/projects/OS\_labs/lab2$ ./main 1 1200000

Probability = 7.12%

Program executed in 325 ms.

**Тест 2:**

luckyabatur@Luckyabatur:~/projects/OS\_labs/lab2$ ./main 12 1200000000

Probability = 7.14%

Program executed in 39189 ms.

**Тест 3:**

luckyabatur@Luckyabatur:~/projects/OS\_labs/lab2$ ./main 1200 999999999

Probability = 7.14%

Program executed in 32796 ms.

**Отображение количества потоков:**

luckyabatur@Luckyabatur:~/projects/OS\_labs/lab2$ ps hH p201460 | wc -l

12

**Strace:**

execve("./main", ["./main", "4", "120000000"], 0x7ffc4caaba08 /\* 36 vars \*/) = 0

brk(NULL) = 0x5577600cd000

mmap(NULL, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7f1f9441a000

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

fstat(3, {st\_mode=S\_IFREG|0644, st\_size=23499, ...}) = 0

mmap(NULL, 23499, PROT\_READ, MAP\_PRIVATE, 3, 0) = 0x7f1f94414000

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\0\220\243\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

fstat(3, {st\_mode=S\_IFREG|0755, st\_size=2125328, ...}) = 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, 2170256, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) = 0x7f1f94202000

mmap(0x7f1f9422a000, 1605632, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x28000) = 0x7f1f9422a000

mmap(0x7f1f943b2000, 323584, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1b0000) = 0x7f1f943b2000

mmap(0x7f1f94401000, 24576, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1fe000) = 0x7f1f94401000

mmap(0x7f1f94407000, 52624, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_ANONYMOUS, -1, 0) = 0x7f1f94407000

close(3) = 0

mmap(NULL, 12288, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7f1f941ff000

arch\_prctl(ARCH\_SET\_FS, 0x7f1f941ff740) = 0

set\_tid\_address(0x7f1f941ffa10) = 219542

set\_robust\_list(0x7f1f941ffa20, 24) = 0

rseq(0x7f1f94200060, 0x20, 0, 0x53053053) = 0

mprotect(0x7f1f94401000, 16384, PROT\_READ) = 0

mprotect(0x55775e844000, 4096, PROT\_READ) = 0

mprotect(0x7f1f94452000, 8192, PROT\_READ) = 0

prlimit64(0, RLIMIT\_STACK, NULL, {rlim\_cur=8192\*1024, rlim\_max=RLIM64\_INFINITY}) = 0

munmap(0x7f1f94414000, 23499) = 0

getrandom("\x3b\xc2\xe0\xfd\x89\x8e\x90\xf4", 8, GRND\_NONBLOCK) = 8

brk(NULL) = 0x5577600cd000

brk(0x5577600ee000) = 0x5577600ee000

rt\_sigaction(SIGRT\_1, {sa\_handler=0x7f1f9429b520, sa\_mask=[], sa\_flags=SA\_RESTORER|SA\_ONSTACK|SA\_RESTART|SA\_SIGINFO, sa\_restorer=0x7f1f94247320}, 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) = 0x7f1f939fe000

mprotect(0x7f1f939ff000, 8388608, PROT\_READ|PROT\_WRITE) = 0

rt\_sigprocmask(SIG\_BLOCK, ~[], [QUIT], 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=0x7f1f941fe990, parent\_tid=0x7f1f941fe990, exit\_signal=0, stack=0x7f1f939fe000, stack\_size=0x7fff80, tls=0x7f1f941fe6c0}strace: Process 219543 attached**

**=> {parent\_tid=[219543]}, 88) = 219543**

[pid 219543] rseq(0x7f1f941fefe0, 0x20, 0, 0x53053053 <unfinished ...>

[pid 219542] rt\_sigprocmask(SIG\_SETMASK, [QUIT], <unfinished ...>

[pid 219543] <... rseq resumed>) = 0

[pid 219542] <... rt\_sigprocmask resumed>NULL, 8) = 0

[pid 219543] set\_robust\_list(0x7f1f941fe9a0, 24 <unfinished ...>

[pid 219542] mmap(NULL, 8392704, PROT\_NONE, MAP\_PRIVATE|MAP\_ANONYMOUS|MAP\_STACK, -1, 0 <unfinished ...>

[pid 219543] <... set\_robust\_list resumed>) = 0

[pid 219542] <... mmap resumed>) = 0x7f1f931fd000

[pid 219543] rt\_sigprocmask(SIG\_SETMASK, [QUIT], <unfinished ...>

[pid 219542] mprotect(0x7f1f931fe000, 8388608, PROT\_READ|PROT\_WRITE <unfinished ...>

[pid 219543] <... rt\_sigprocmask resumed>NULL, 8) = 0

[pid 219542] <... mprotect resumed>) = 0

[pid 219542] rt\_sigprocmask(SIG\_BLOCK, ~[], [QUIT], 8) = 0

[pid 219542] **clone3({flags=CLONE\_VM|CLONE\_FS|CLONE\_FILES|CLONE\_SIGHAND|CLONE\_THREAD|CLONE\_SYSVSEM|CLONE\_SETTLS|CLONE\_PARENT\_SETTID|CLONE\_CHILD\_CLEARTID, child\_tid=0x7f1f939fd990, parent\_tid=0x7f1f939fd990, exit\_signal=0, stack=0x7f1f931fd000, stack\_size=0x7fff80, tls=0x7f1f939fd6c0}strace: Process 219544 attached**

**=> {parent\_tid=[219544]}, 88) = 219544**

[pid 219544] rseq(0x7f1f939fdfe0, 0x20, 0, 0x53053053 <unfinished ...>

[pid 219542] rt\_sigprocmask(SIG\_SETMASK, [QUIT], <unfinished ...>

[pid 219544] <... rseq resumed>) = 0

[pid 219542] <... rt\_sigprocmask resumed>NULL, 8) = 0

[pid 219544] set\_robust\_list(0x7f1f939fd9a0, 24 <unfinished ...>

[pid 219542] mmap(NULL, 8392704, PROT\_NONE, MAP\_PRIVATE|MAP\_ANONYMOUS|MAP\_STACK, -1, 0 <unfinished ...>

[pid 219544] <... set\_robust\_list resumed>) = 0

[pid 219542] <... mmap resumed>) = 0x7f1f929fc000

[pid 219544] rt\_sigprocmask(SIG\_SETMASK, [QUIT], <unfinished ...>

[pid 219542] mprotect(0x7f1f929fd000, 8388608, PROT\_READ|PROT\_WRITE <unfinished ...>

[pid 219544] <... rt\_sigprocmask resumed>NULL, 8) = 0

[pid 219542] <... mprotect resumed>) = 0

[pid 219542] rt\_sigprocmask(SIG\_BLOCK, ~[], [QUIT], 8) = 0

[pid 219542] **clone3({flags=CLONE\_VM|CLONE\_FS|CLONE\_FILES|CLONE\_SIGHAND|CLONE\_THREAD|CLONE\_SYSVSEM|CLONE\_SETTLS|CLONE\_PARENT\_SETTID|CLONE\_CHILD\_CLEARTID, child\_tid=0x7f1f931fc990, parent\_tid=0x7f1f931fc990, exit\_signal=0, stack=0x7f1f929fc000, stack\_size=0x7fff80, tls=0x7f1f931fc6c0}strace: Process 219545 attached**

**=> {parent\_tid=[219545]}, 88) = 219545**

[pid 219545] rseq(0x7f1f931fcfe0, 0x20, 0, 0x53053053 <unfinished ...>

[pid 219542] rt\_sigprocmask(SIG\_SETMASK, [QUIT], <unfinished ...>

[pid 219545] <... rseq resumed>) = 0

[pid 219542] <... rt\_sigprocmask resumed>NULL, 8) = 0

[pid 219545] set\_robust\_list(0x7f1f931fc9a0, 24 <unfinished ...>

**[pid 219542] futex(0x7f1f941fe990, FUTEX\_WAIT\_BITSET|FUTEX\_CLOCK\_REALTIME, 219543, NULL, FUTEX\_BITSET\_MATCH\_ANY <unfinished ...>**

[pid 219545] <... set\_robust\_list resumed>) = 0

[pid 219545] rt\_sigprocmask(SIG\_SETMASK, [QUIT], NULL, 8) = 0

[pid 219545] rt\_sigprocmask(SIG\_BLOCK, ~[RT\_1], NULL, 8) = 0

[pid 219545] madvise(0x7f1f929fc000, 8368128, MADV\_DONTNEED) = 0

[pid 219545] exit(0) = ?

[pid 219545] +++ exited with 0 +++

[pid 219543] rt\_sigprocmask(SIG\_BLOCK, ~[RT\_1], NULL, 8) = 0

[pid 219543] madvise(0x7f1f939fe000, 8368128, MADV\_DONTNEED) = 0

[pid 219543] exit(0) = ?

[pid 219542] <... futex resumed>) = 0

[pid 219543] +++ exited with 0 +++

**[pid 219542] futex(0x7f1f939fd990, FUTEX\_WAIT\_BITSET|FUTEX\_CLOCK\_REALTIME, 219544, NULL, FUTEX\_BITSET\_MATCH\_ANY <unfinished ...>**

[pid 219544] rt\_sigprocmask(SIG\_BLOCK, ~[RT\_1], NULL, 8) = 0

[pid 219544] madvise(0x7f1f931fd000, 8368128, MADV\_DONTNEED) = 0

[pid 219544] exit(0) = ?

[pid 219542] <... futex resumed>) = 0

[pid 219544] +++ exited with 0 +++

fstat(1, {st\_mode=S\_IFCHR|0620, st\_rdev=makedev(0x88, 0x2), ...}) = 0

**write(1, "Probability = 7.14%\n", 20Probability = 7.14%**

**) = 20**

**write(1, "Program executed in 10945 ms.\n", 30Program executed in 10945 ms.**

**) = 30**

exit\_group(0) = ?

+++ exited with 0 +++

**Анализ ускорения и эффективности**

1. 1200000 раундов.

|  |  |  |  |
| --- | --- | --- | --- |
| Количество потоков | Время выполнения (мс) | Ускорение | Эффективность |
| 1 | 316 | 1 | 1 |
| 2 | 173 | 1,826589595 | 0,913294798 |
| 3 | 112 | 2,821428571 | 0,94047619 |
| 4 | 88 | 3,590909091 | 0,897727273 |
| 5 | 71 | 4,450704225 | 0,890140845 |
| 6 | 56 | 5,642857143 | 0,94047619 |
| 7 | 55 | 5,745454545 | 0,820779221 |
| 8 | 48 | 6,583333333 | 0,822916667 |
| 9 | 43 | 7,348837209 | 0,816537468 |
| 10 | 40 | 7,9 | 0,79 |
| 11 | 37 | 8,540540541 | 0,776412776 |
| 12 | 39 | 8,102564103 | 0,675213675 |
| 13 | 42 | 7,523809524 | 0,578754579 |
| 14 | 47 | 6,723404255 | 0,480243161 |

1. 300000000 раундов.

|  |  |  |  |
| --- | --- | --- | --- |
| Количество потоков | Время выполнения (мс) | Ускорение | Эффективность |
| 1 | 81946 | 1 | 1 |
| 2 | 42336 | 1,935610355 | 0,967805178 |
| 3 | 27740 | 2,95407354 | 0,98469118 |
| 4 | 22022 | 3,721097085 | 0,930274271 |
| 5 | 19069 | 4,297341234 | 0,859468247 |
| 6 | 15167 | 5,402914222 | 0,900485704 |
| 7 | 14449 | 5,671395944 | 0,810199421 |
| 8 | 11833 | 6,925209161 | 0,865651145 |
| 9 | 10700 | 7,658504673 | 0,850944964 |
| 10 | 10265 | 7,983049196 | 0,79830492 |
| 11 | 9886 | 8,289095691 | 0,753554154 |
| 12 | 8939 | 9,167244658 | 0,763937055 |
| 13 | 9282 | 8,82848524 | 0,679114249 |
| 14 | 9701 | 8,447170395 | 0,603369314 |

**Ускорение** показывает во сколько раз применение параллельного алгоритма уменьшает время решения задачи по сравнению с последовательным алгоритмом. Ускорение определяется величиной SN=T1/TN, где Т1 - время выполнения на одном потоке, TN - время выполнения на N потоках.

**Эффективность** - величина EN = SN/N, где SN - ускорение, N - количество используемых потоков.

**Вывод**

Первый раз поработал с многопоточностью смог за её счёт значительно ускорить время выполнения программы. Происходит значительный прирост в скорости выполнения при увеличении количества физических ядер (в моём случае – 6), ещё немного скорость растёт в процессе достижения количества логических ядер (12). Далее показатели падают, т.к. параллельно все потоки уже выполняться не могут, и начинают тратиться ресурсы на их переключение.