

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: Гуляев А.П.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 5.11.24

Москва, 2024

Постановка задачи

Вариант 2.

Отсортировать массив целых чисел при помощи параллельного алгоритма быстрой сортировки.

Общий метод и алгоритм решения

Использованные системные вызовы:

- `pid_t fork(void)`; – создание дочернего процесса
- `pid_t wait(int)` - ожидание завершения дочерних процессов
- `key_t ftok (const char *, int)` – создание ключа System V IPC
- `int shmget(key_t, size_t, int)` – получение дескриптора (создание) разделяемого сегмента памяти
- `void *shmat(int, const void*, int)` – внесение разделяемого сегмента памяти в пространство имен процесса
- `int shmdt(const void*)` - удаление разделяемого сегмента памяти из пространства имен процесса
- `int shmctl(int, int, struct shmid_ds *)` - удаление разделяемого сегмента памяти

Описание работы программы:

1. Инициализация и обработка аргументов командной строки:

- Программа ожидает один аргумент командной строки — максимальное количество параллельных процессов (поток).
- Если аргумент не передан, программа выводит сообщение об ошибке и завершает работу.

2. Открытие файла и чтение данных:

- Открывается файл `test.txt`, из которого считывается количество чисел `n`.
- Программа выделяет память для массива `array` в общей памяти, используя системные вызовы `ftok`, `shmget` и `shmat`.
- С помощью `ftok` генерируется уникальный ключ для сегмента общей памяти.
- Вся память, включая последний элемент массива (для отслеживания числа работающих потоков), выделяется с помощью `shmget`.

3. Чтение массива:

- Считывается массив чисел из файла `test.txt` в общий сегмент памяти, куда привязана переменная `array`.

4. Инициализация количества потоков:

- Устанавливается указатель `running_threads` на последний элемент выделенной общей памяти, чтобы отслеживать число запущенных процессов.

Изначально в `running_threads` устанавливается значение 1, так как родительский процесс уже запущен.

5. Рекурсивная сортировка с поддержкой многопроцессности:

- Вызов функции `my_qsort` начинает сортировку массива.
- Внутри функции `my_qsort` для части массива выбирается опорный элемент `seed`.
- Затем массив разделяется на две части: элементы, меньшие `seed`, и элементы, большие или равные `seed`.
- Если количество запущенных потоков не превышает максимальное количество потоков и размер подмассива достаточно велик, создается новый процесс с помощью `fork`.
- Новый процесс рекурсивно сортирует одну часть массива, а родительский процесс — другую.
- Если лимит потоков достигнут или размер подмассива недостаточно велик, оба подмассива сортируются в рамках текущего процесса.

6. Ожидание завершения всех потоков:

- Родительский процесс ожидает завершения всех дочерних процессов, используя `wait`, пока значение в `running_threads` не уменьшится до 1.

7. Запись отсортированных данных в файл:

- По завершении сортировки открывается файл `test_out.txt`, и отсортированные данные записываются в него.

8. Отключение и удаление общей памяти:

- Программа отключается от общей памяти с помощью `shmdt`, удаляет сегмент с помощью `shmctl`, а затем завершает работу.

Код программы

```
#include <stdio.h>
```

```
#include <stdlib.h>
```

```
#include <sys/types.h>
```

```
#include <sys/wait.h>
```

```
#include <unistd.h>
```

```
#include <fcntl.h>
```

```
#include <sys/shm.h>
```

```
#include <sys/ipc.h>
```

```

#include <sys/time.h>

struct timeval tv1, tv2, dtv;

struct timezone tz;

void time_start() { gettimeofday(&tv1, &tz); }

long time_stop()
{ gettimeofday(&tv2, &tz);
  dtv.tv_sec = tv2.tv_sec - tv1.tv_sec;
  dtv.tv_usec = tv2.tv_usec - tv1.tv_usec;
  if(dtv.tv_usec < 0) { dtv.tv_sec--; dtv.tv_usec += 1000000; }
  return dtv.tv_sec * 1000 + dtv.tv_usec / 1000;
}

```

```

void swap(long long *a, long long *b) {
  long long buff = *a;
  *a = *b;
  *b = buff;
}

```

```

void my_qsort(long long *begin, long long *end, int max_threads_count, long long
*running_threads, long long *array_begin) {
  if (end - begin < 2) {
    return;
  }
  long long *seed = end - 1;
  long long *left = begin;
  long long *right = end - 2;

  while (left < right) {
    if ((*left >= *seed) && (*right < *seed)) {

```

```

        swap(left, right);
    }
    while (*left < *seed) {
        ++left;
    }
    while (right >= begin && *right >= *seed) {
        --right;
    }
}

if (*left > *seed) {
    swap(left, seed);
}
seed = left;

while (*left == *seed) {
    ++left;
}
++right;

if ((*running_threads < max_threads_count) && (end - begin > (int)1e5)) {
    (*running_threads)++;
    pid_t child = fork();
    switch (child)
    {
    case -1:
        printf("fork error\n");
        (*running_threads)--;
        break;
    case 0:
        my_qsort(left, end, max_threads_count, running_threads, array_begin);
        (*running_threads)--;
    }
}

```

```

        if(shmdt(array_begin) < 0) {
            printf("shmdt error\n");
        }
        exit(0);
        break;

    default:
        my_qsort(begin, right, max_threads_count, running_threads, array_begin);
        break;
    }
} else {
    my_qsort(begin, right, max_threads_count, running_threads, array_begin);
    my_qsort(left, end, max_threads_count, running_threads, array_begin);
}

return;
}

```

```

int main(int argc, char* argv[]) {
    if (argc != 2) {
        printf("wrong number of args\n");
        return -1;
    }

```

```

    int max_threads_count = atoi(argv[1]);

```

```

    FILE* file;
    if ((file = fopen("test.txt", "r")) == NULL) {
        printf("file error\n");
        return -1;
    }

```

```
long long n;
```

```
fscanf(file, "%lld", &n);
```

```
long long *array;
```

```
int shmid;
```

```
char pathname[] = "main.c";
```

```
key_t key;
```

```
if((key = ftok(pathname, 0)) < 0) {
```

```
    printf("ftok error\n");
```

```
    fclose(file);
```

```
    return -1;
```

```
}
```

```
if((shmid = shmget(key, (n + 1) * sizeof(long long), 0666|IPC_CREAT|IPC_EXCL)) < 0) {
```

```
    printf("shmget error\n");
```

```
    fclose(file);
```

```
    return -1;
```

```
}
```

```
if((array = (long long *)shmat(shmid, NULL, 0)) == (long long *)(-1)) {
```

```
    printf("shmat error\n");
```

```
    fclose(file);
```

```
    return -1;
```

```
}
```

```
for (long long i = 0; i < n; ++i) {
```

```
    fscanf(file, "%lld", &array[i]);
```

```
}
```

```
fclose(file);
```

```
long long *running_threads = &(array[n]);
```

```

(*running_threads) = 1;

time_start();
printf("%d started...\n", getpid());
my_qsort(array, array + n, max_threads_count, running_threads, array);
while (*running_threads > 1) {
    wait(0);
}
printf("%d completed in %ldms\n", getpid(), time_stop());

if ((file = fopen("test_out.txt", "w")) == NULL) {
    printf("file error\n");
    return -1;
}

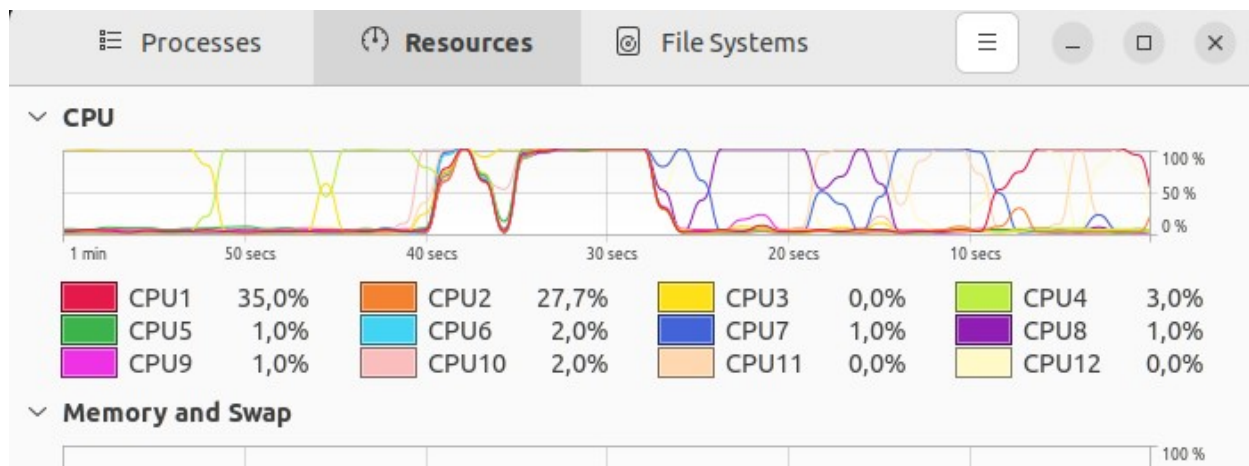
for (long long i = 0; i < n; ++i) {
    fprintf(file, "%lld\n", array[i]);
}
putc('\n', file);
fclose(file);

if(shmdt(array) < 0) {
    printf("shmdt error\n");
    return -1;
}
if (shmctl(shmid, 0, NULL) < 0) {
    printf("shmctl error\n");
    return -1;
}
return 0;
}

```


Протокол работы программы

```
PROBLEMS  DEBUG CONSOLE  OUTPUT  TERMINAL  PORTS
• npabwa@npabwa-Vivobook-ASUSLaptop-M6500QH-M6500QH:~/Desktop/MAI_OS/lab02/src$ ./sol.out 1
6816 started...
6816 completed in 56583ms
• npabwa@npabwa-Vivobook-ASUSLaptop-M6500QH-M6500QH:~/Desktop/MAI_OS/lab02/src$ ./sol.out 24
6927 started...
6927 completed in 14453ms
○ npabwa@npabwa-Vivobook-ASUSLaptop-M6500QH-M6500QH:~/Desktop/MAI_OS/lab02/src$
```



Тестирование:

Написал 2 python-скрипта, один из которых генерирует набор чисел, а второй проверяет выходной файл на предмет отсортированности. Протестировал поведение программы на различных наборах данных, проверил корректность алгоритма. Скрипты ниже:

generate.py:

```
import random

n = int(5 * 1e8)

a = int(1e7)

with open("/home/npabwa/Desktop/MAI_OS/lab02/src/test.txt", 'w') as file:

    file.write(f"{n}\n")

    for i in range(n):

        file.write(f"{random.randint(-a, a)} ")

    file.write("\n")
```

checker.py

with open("/home/npabwa/Desktop/MAI_OS/lab02/src/test_out.txt", 'r') as file:

```
    prev = int(file.readline().strip(" \n"))
```

```
    for i in range(int(5 * 1e8) - 1):
```

```
        n = int(file.readline().strip(" \n"))
```

```
        if (n < prev):
```

```
            print(f"not sorted, line {i}, prev = {prev}, n = {n}")
```

```
            break
```

```
        prev = n
```

```
    else:
```

```
        print("sorted")
```

Вывод

Реализовал алгоритм многопоточной быстрой сортировки.

Время выполнения от количества потоков зависит следующим образом (тестировал на случайно сгенерированных данных, $5 \cdot 10^8$, процессор 6 ядер 12 потоков):

1 поток – 56.7 с

2 потока – 41.6 с

4 потока – 22.0 с

8 потоков – 14.0 с

12 потоков – 13.6 с

16 потоков – 13.9 с

24 потока – 13.8 с

При увеличении количества потоков в 2 раза, время исполнения уменьшается в 1.5 – 2 раза (зависит от входных данных)