



Министерство науки и высшего образования Российской Федерации
Мытищинский филиал
Федерального государственного автономного образовательного
учреждение
высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»
(МФ МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ Космический

КАФЕДРА К3

отчет

К ЛАБОРАТОРНОЙ РАБОТЕ

№ 2

по дисциплине

«ОПЕРАЦИОННЫЕ СИСТЕМЫ»

Студент К3-76Б
(Группа)

В. Д. Чернов
(И.О.Фамилия)

Преподаватель

А. В. Чернышов
(И.О.Фамилия)

Задание к лабораторной работе

Написать на языке си программу выполняющую моделирование работы дисковой подсистемы. (Реально обращения к диску выполняться не должны!). Исходные данные следующие: геометрия диска 500 дорожек(цилиндров), 4 поверхности\головки, 16 секторов на поверхности, скорость вращения 10к об\мин, время чтения одного сектора равно времени прохождения сектора под головкой(время ожидания сектора вычислять исходя из номера запрошенного сектора, числа пересекаемых головкой дорожек и скорости вращения диска) время записи одного сектора складывается из времени прохождения сектора под головкой для собственно записи и повторного прохождения для верификации.

РЕШЕНИЯ ЗАДАНИЯ

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>

#define MODEL_TIME 300000 // Время моделирования (мс)
#define CYLINDERS 500 // Количество цилиндров
#define HEADS 4 // Количество головок
#define SECTORS 16 // Количество секторов
#define SEEK_TIME_PER_TRACK 0.5 // Время перемещения на один цилиндр
                           // (мс)

#define ROTATIONAL_SPEED 10000 // Скорость вращения диска (об/мин)
#define ROTATIONAL_DELAY (60.0 / ROTATIONAL_SPEED * 1000) // Время
                           // на оборот (мс)

#define SECTOR_PASS_TIME (ROTATIONAL_DELAY / SECTORS) // Время
                           // прохождения одного сектора (мс)
```

```
typedef struct {
    int cylinder;
    int head;
    int sector;
    int isRead;
    int sectorCount;
    double arrivalTime;
    int processed;
} Request;
```

```
typedef struct {
    double minTime;
    double maxTime;
    double totalTime;
    double idleTime;
    int maxQueueLength;
    double* serviceTimes;
```

```
int serviceCount;
} Stats;

int generateRequests(Request **requests, double tmax) {
    srand(time(0));
    double currentTime = 0.0;
    int i = 0;

    int max_num = (int)(MODEL_TIME / tmax) * 2;

    *requests = (Request *)malloc(max_num * sizeof(Request));

    if (*requests == NULL) {
        printf("Ошибка выделения памяти!\n");
        exit(1);
    }

    while (currentTime < MODEL_TIME && i < max_num) {
        double interval = ((double)rand() / RAND_MAX) * tmax;
        currentTime += interval;

        if (currentTime < MODEL_TIME) {
            (*requests)[i].cylinder = rand() % CYLINDERS;
            (*requests)[i].head = rand() % HEADS;
            (*requests)[i].sector = rand() % SECTORS;
            (*requests)[i].isRead = rand() % 2;
            (*requests)[i].sectorCount = 1;
            (*requests)[i].arrivalTime = currentTime;
            (*requests)[i].processed = 0;
            i++;
        }
    }

    return i;
}

void calculateStatistics(Stats *stats) {
```

```

if (stats->serviceCount == 0) {
    printf("Нет обработанных запросов для расчета статистики.\n");
    return;
}

double mean = stats->totalTime / stats->serviceCount;
double variance = 0.0;

for (int i = 0; i < stats->serviceCount; i++) {
    double diff = stats->serviceTimes[i] - mean;
    variance += diff * diff;
}

variance /= stats->serviceCount;
double stdDev = sqrt(variance);

printf("Минимальное время обслуживания запроса: %.2f мс\n", stats-
>minTime);
printf("Максимальное время обслуживания запроса: %.2f мс\n", stats-
>maxTime);
printf("Среднее время обслуживания запроса: %.2f мс\n", mean);
printf("Среднеквадратическое отклонение: %.2f мс\n", stdDev);
printf("Максимальная длина очереди: %d\n", stats->maxQueueLength);
printf("Суммарное время простоя дисковой подсистемы: %.2f мс\n", stats-
>idleTime);
}

double process_request(int currentCylinder, int currentSector, Request request,
                      Stats *stats, double *currentTime) {
    int seekDistance = abs(request.cylinder - currentCylinder);
    double seekTime = seekDistance * SEEK_TIME_PER_TRACK;
    double sectorWaitTime = (double)(abs(request.sector - currentSector)) *
        SECTOR_PASS_TIME;

    double timeForRequest = seekTime + sectorWaitTime + request.sectorCount *
        SECTOR_PASS_TIME;
    if (!request.isRead) {

```

```

        timeForRequest += request.sectorCount * SECTOR_PASS_TIME;
    }

    if (timeForRequest < stats->minTime) stats->minTime = timeForRequest;
    if (timeForRequest > stats->maxTime) stats->maxTime = timeForRequest;

    stats->serviceTimes[stats->serviceCount] = timeForRequest;
    stats->serviceCount++;

    *currentTime += timeForRequest;

    return timeForRequest;
}

int getMaxQueueLength(Request requests[], int num, double currentTime, double
timeForRequest) {
    int queueLength = 0;

    for (int i = 0; i < num; i++) {
        if (requests[i].arrivalTime >= currentTime && requests[i].arrivalTime <=
currentTime + timeForRequest)
            queueLength++;
    }
    return queueLength;
}

double fifoStrategy(Request requests[], int num, Stats *stats) {
    int currentCylinder = 0;
    int currentSector = 0;
    double totalSeekTime = 0.0;
    double currentTime = 0.0;

    stats->minTime = 1000000;
    stats->maxTime = 0;
    stats->idleTime = 0;
    stats->serviceCount = 0;
    stats->maxQueueLength = 0;
}

```

```

stats->totalTime = 0;

stats->serviceTimes = malloc(sizeof(double) * num);

if (stats->serviceTimes == NULL) {
    printf("Ошибка выделения памяти для serviceTimes!\n");
    exit(1);
}

for (int i = 0; i < num; i++) {
    if (currentTime < requests[i].arrivalTime) {
        currentTime = requests[i].arrivalTime;
    }

    if (currentTime >= MODEL_TIME) {
        stats->maxQueueLength = (int)fmax(stats->maxQueueLength, num - i);
        break;
    }
}

double timeForRequest = process_request(currentCylinder, currentSector,
requests[i], stats, &currentTime);
totalSeekTime += timeForRequest;

int queueLen = getMaxQueueLength(requests, num, currentTime,
timeForRequest);
if (queueLen > stats->maxQueueLength) {
    stats->maxQueueLength = queueLen;
}

currentCylinder = requests[i].cylinder;
currentSector = requests[i].sector;
}

stats->idleTime = fmax(0.0, MODEL_TIME - totalSeekTime);
stats->totalTime = totalSeekTime;

return totalSeekTime;
}

```

```
double fscanStrategy(Request requests[], int num, Stats *stats) {  
    double currentTime = 0.0;  
    int currentCylinder = 0;  
    int currentSector = 0;  
    double totalSeekTime = 0.0;  
  
    int direction = 1;  
  
    Request *queue = malloc(num * sizeof(Request));  
    for (int i = 0; i < num; i++) {  
        queue[i] = requests[i];  
        queue[i].processed = 0;  
    }  
  
    stats->minTime = 1000000;  
    stats->maxTime = 0;  
    stats->idleTime = 0;  
    stats->maxQueueLength = 0;  
    stats->serviceCount = 0;  
    stats->totalTime = 0;  
    stats->serviceTimes = malloc(sizeof(double) * num);  
  
    if (stats->serviceTimes == NULL) {  
        printf("Ошибка выделения памяти для serviceTimes!\n");  
        free(queue);  
        exit(1);  
    }  
  
    int processed = 0;  
  
    while (processed < num && currentTime < MODEL_TIME) {  
        int freezeCount = 0;
```

```

for (int i = 0; i < num; i++) {
    if (!queue[i].processed && queue[i].arrivalTime <= currentTime)
        freezeCount++;
}

if (freezeCount == 0) {
    double nextTime = MODEL_TIME;
    for (int i = 0; i < num; i++) {
        if (!queue[i].processed && queue[i].arrivalTime < nextTime)
            nextTime = queue[i].arrivalTime;
    }
    if (nextTime != MODEL_TIME)
        currentTime = nextTime;
    else
        break;
    continue;
}

int servedSomething = 1;
while (servedSomething) {
    servedSomething = 0;

    int bestIndex = -1;
    int bestCylinder = (direction == 1 ? CYLINDERS + 1 : -1);

    for (int i = 0; i < num; i++) {
        if (queue[i].processed || queue[i].arrivalTime > currentTime)
            continue;

        if (direction == 1) {
            if (queue[i].cylinder >= currentCylinder &&
                queue[i].cylinder < bestCylinder)
            {
                bestCylinder = queue[i].cylinder;
                bestIndex = i;
            }
        } else {

```

```

        if (queue[i].cylinder <= currentCylinder &&
            queue[i].cylinder > bestCylinder)
        {
            bestCylinder = queue[i].cylinder;
            bestIndex = i;
        }
    }

    if (bestIndex != -1) {
        double timeForRequest = process_request(
            currentCylinder, currentSector, queue[bestIndex], stats, &currentTime
        );

        currentCylinder = queue[bestIndex].cylinder;
        currentSector = queue[bestIndex].sector;

        queue[bestIndex].processed = 1;
        processed++;
        totalSeekTime += timeForRequest;

        servedSomething = 1;
    }
}

direction = 1 - direction;
}

stats->idleTime = fmax(0.0, MODEL_TIME - totalSeekTime);
stats->totalTime = totalSeekTime;

free(queue);
return totalSeekTime;
}

void saveDataToFile(const char *filename, double *serviceTimes, int count) {
    FILE *file = fopen(filename, "w");

```

```
if (file == NULL) {
    printf("Ошибка открытия файла %s для записи!\n", filename);
    exit(1);
}

for (int i = 0; i < count; i++) {
    fprintf(file, "%f\n", serviceTimes[i]);
}

fclose(file);
printf("\nДанные сохранены в файл: %s\n", filename);
}

int main() {
    Request *requests = NULL;
    Stats stats;
    double tmax_values[] = {5000, 500, 50};

    for (int i = 0; i < 3; i++) {
        double tmax = tmax_values[i];
        printf("\nРезультаты для TMAX = %.0f мс\n", tmax);

        int numRequests = generateRequests(&requests, tmax);
        printf("Сгенерировано %d запросов.\n", numRequests);

        if (numRequests == 0) {
            printf("Нет запросов для обработки.\n");
            continue;
        }

        printf("\nИспользуется стратегия FIFO\n");
        fifoStrategy(requests, numRequests, &stats);
        calculateStatistics(&stats);

        char fifoFilename[50];
        snprintf(fifoFilename, sizeof(fifoFilename), "fifo_tmax_%.0f.csv", tmax);
```

```
saveDataToFile(fifoFilename, stats.serviceTimes, stats.serviceCount);
free(stats.serviceTimes);

printf("\nИспользуется стратегия F-SCAN\n");
fscanStrategy(requests, numRequests, &stats);
calculateStatistics(&stats);

char fscanFilename[50];
snprintf(fscanFilename, sizeof(fscanFilename), "fscan_tmax_%.0f.csv", tmax);
saveDataToFile(fscanFilename, stats.serviceTimes, stats.serviceCount);
free(stats.serviceTimes);

free(requests);
requests = NULL;
}

return 0;
}
```

Результат работы

Результаты для ТМАХ = 5000 мс
Сгенерировано 118 запросов.

Используется стратегия FIFO

Минимальное время обслуживания запроса: 1.25 мс
Максимальное время обслуживания запроса: 234.12 мс
Среднее время обслуживания запроса: 87.52 мс
Среднеквадратическое отклонение: 64.16 мс
Максимальная длина очереди: 1
Суммарное время простоя дисковой подсистемы: 289672.25 мс

Данные сохранены в файл: fifo_tmax_5000.csv

Используется стратегия F-SCAN

Минимальное время обслуживания запроса: 1.25 мс
Максимальное время обслуживания запроса: 234.12 мс
Среднее время обслуживания запроса: 87.52 мс
Среднеквадратическое отклонение: 64.16 мс
Максимальная длина очереди: 0
Суммарное время простоя дисковой подсистемы: 289672.25 мс

Данные сохранены в файл: fscan_tmax_5000.csv

Результаты для ТМАХ = 500 мс
Сгенерировано 1197 запросов.

Используется стратегия FIFO

Минимальное время обслуживания запроса: 1.12 мс
Максимальное время обслуживания запроса: 250.12 мс
Среднее время обслуживания запроса: 84.89 мс
Среднеквадратическое отклонение: 59.08 мс
Максимальная длина очереди: 4
Суммарное время простоя дисковой подсистемы: 198389.62 мс

Данные сохранены в файл: fifo_tmax_500.csv

Используется стратегия F-SCAN

Минимальное время обслуживания запроса: 1.12 мс
Максимальное время обслуживания запроса: 250.12 мс
Среднее время обслуживания запроса: 83.95 мс
Среднеквадратическое отклонение: 58.67 мс
Максимальная длина очереди: 0
Суммарное время простоя дисковой подсистемы: 199514.38 мс

Данные сохранены в файл: fscan_tmax_500.csv

Результаты для ТМАХ = 50 мс
Сгенерировано 12000 запросов.

Используется стратегия FIFO

Минимальное время обслуживания запроса: 0.38 мс
Максимальное время обслуживания запроса: 250.12 мс
Среднее время обслуживания запроса: 85.18 мс
Среднеквадратическое отклонение: 58.36 мс
Максимальная длина очереди: 8478
Суммарное время простоя дисковой подсистемы: 9.12 мс

Данные сохранены в файл: fifo_tmax_50.csv

Используется стратегия F-SCAN

Минимальное время обслуживания запроса: 0.38 мс

Максимальное время обслуживания запроса: 217.12 мс

Среднее время обслуживания запроса: 24.93 мс

Среднеквадратическое отклонение: 27.31 мс

Максимальная длина очереди: 0

Суммарное время простоя дисковой подсистемы: 797.62 мс

Данные сохранены в файл: fscan_tmax_50.csv

