

Министерство науки и высшего образования Российской Федерации
Новосибирский государственный технический университет

Управление ресурсами в вычислительных системах
Лабораторная работа №2

Факультет: прикладной математики и информатики
Группа: ПМ-13
Студенты: Исакин Д. А.
Вострецова Е. В.
Преподаватели: Стасышин В. М.
Сивак М. А.

Новосибирск
2024

1. Условие (Вариант №10)

Разработать программу, вычисляющую интеграл на отрезке $[A;B]$ от функции $f(x) = \frac{e^x - e^{-x}}{2}$ методом трапеций, разбивая интервал на K равных отрезков. Для нахождения e^x и e^{-x} программа должна породить параллельные процессы, вычисляющие эти значение путём разложения в ряд по формулам вычислительной математики.

Разработать программу, реализующую действия, указанные в задании к лабораторной работе с учётом следующих требований:

- все действия, относящиеся как к родительскому процессу, так и к порожденным процессам, выполняются в рамках одного исполняемого файла;
- обмен данными между процессом-отцом и процессом-потомком предлагается выполнить посредством временного файла: процесс-отец после порождения процесса-потомка постоянно опрашивает временный файл, ожидая появления в нем информации от процесса-потомка;
- если процессов-потомков несколько, и все они подготавливают некоторую информацию для процесса-родителя, каждый из процессов помещает в файл некоторую структурированную запись, при этом в этой структурированной записи содержатся сведения о том, какой процесс посылает запись, и сама подготовленная информация.

2. Анализ задачи

1. Вычисляем значение функции e^{-x} при помощи разложения в ряд

телора по формуле: $\sum_{n=0}^{\infty} \frac{(-1)^n x^n}{n!}$ Данный ряд является рядом

Лейбницевского типа поэтому погрешность можно оценить так:

$|R_n| < b_{n+1}$. Далее учтя, что $e^x = \frac{1}{e^{-x}}$ вычислим значение второй функции.


2. Каждая из функций вычисляется в своем процессе. После этого процесс родитель считывает данные из файла и рассчитывает итоговую функцию на каждом шаге расчета интеграла
3. Расчет интеграла производится по следующей формуле Нбютона-Котесса - формула трапеций:
так как сетка равномерная $x_i = A + i \cdot h$, где $h = \frac{B-A}{K}$ тформула трапеций запишется в следующем виде

$$\int_A^B f(x)dx = h \left(\frac{f_0 + f_K}{2} \sum_{i=1}^{K-1} f_i \right) + E_n(f)$$

Где для погрешность справедлива оценка $E_n(f) = -\frac{f''(\xi)}{12}(B-A)h^2$, где $\xi \in [A, B]$.

3. Используемые программные средства

Функции:

`int fork()` - порождение процесса-потомка 

`FILE *fopen(const char *__restrict __filename, const char *__res1`

`int fclose(FILE *__stream)` - закрытие файла

`void exit(int __status)` - выход из программы с заданным кодом возвра

`int fprintf(FILE *__restrict __stream, const char *__restrict __`

`int fscanf(FILE *__restrict __stream, const char *__restrict __`

`void *memmove(void *__dest, const void *__src, size_t __n)` - Копир

`char *strcat(char *__restrict __dest, const char *__restrict __`

`pid_t fork(void)` - Порождение нового процесса

`pid_t getpid(void)` - получение идентификатора процесса

`pid_t waitpid(pid_t __pid, int *__stat_loc, int __options)` - Ожидан

4. Спецификация

Программа находится в папке /home/daniil/Desktop/WorkSpace/УпРесы/
resource-management-in-computing-systems/lab2/ Чтобы собрать программу
нужно ввести "make all" Чтобы запустить программу, нужно использовать
команду "./main" В результате работы программы, будет показано
значение интегралла рассчитанного от функции $f(x)$ на отрезке от [A;B]
Формат входного файла: A B K

5. Тесты

Тест №1

Описание: Корректные входные данные

| input.txt | Результат работы программы | Истинное значение интеграла |
|-----------|-------------------------------------|------------------------------|
| 0 5 300 | $\int_0^5 f(x)dx \approx 73.211643$ | $\int_0^5 f(x)dx = 73.20995$ |

Тест №2

Описание: Некорректные входные параметры

| input.txt | Результат работы программы |
|-----------|----------------------------|
| 0 а 300 | Error read file: input.txt |

6. Исходный код программы

main.c



```
/* Код программы для лабораторной работы №2 */
#include <math.h>      // fabs()
#include <stdio.h>     // io-functions
#include <sys/types.h> // pid_t
#include <unistd.h>    // fork(), sleep(), usleep()
#include <stdlib.h>    // fprintf(), fscanf()
#include <string.h>
#include <inttypes.h>
#include <sys/wait.h>

/* system cmd */
char *_touch = "touch ";
char *_rm = "rm ";
char *_2_dev_null = " 2> /dev/null";
/*****/

/* ERROR CODE */
#define OK 0 // OK
#define DOUBLE_OVERFLOW 1 // Переполнение типа double
#define INCORRECT_GRID_INPUT_DATA 2 // Некорретные данные для сетки
#define FILE_OPEN_ERROR 3 // Файл не возможно открыть
#define FILE_READ_ERROR 4 // Ошибка при чтении файла
#define FUNCTION_CALCULATION_ERROR 5 // Ошибка при расчете функции
/*****/

/* System data */
const char *InputFile = "input.txt";
const char *InterProcessFileForCommunication1 = "InterProcess1.txt"
const char *InterProcessFileForCommunication2 = "InterProcess2.txt"
FILE *_fd[2];
pid_t _pid1, _pid2;
/*****/

/* Math struct and var */
const double eps = 1e-15; // машинный ноль

/*
@details: Сетка для расчетной области шаг по области равномерный
*/
struct _Grid
{
    int32_t K; // Количество шагов
    double A; // Начало сетки
    double B; // Конец
    double step; // шаг на отрезке
};

/*****/

/* MyMath */
```

```

/* Структура для представления функции */
struct _f_x
{
    pid_t pid1; // Номер первого процесса
    double f1;  // exp(x)

    pid_t pid2; // Номер 2-ого процесса
    double f2;  // exp(-x)
    double f;   // (f1-f2)/2
} fx;

/*
    @param:
        double x
    @return: double
    @details: Расчет экспоненты разложением в ряд тейлора
*/
double _exp(double x);

/*
    @param: double x
    @return: void
    @result: Запускает 2 процесса рассчитывает функцию и в конце обр
        При этом в поле fx.f будет результат расчета целевой ф
        Обработка ошибок происходит внутри в случае ошибки вы
*/
void CalcFunction_fx(double x);

/*
    @param: const struct _Grid *Grid - сетка
    @result: double - результат расчета интеграла
    @result: -
*/
double IntegrateTrapetcoid(const struct _Grid *Grid);
/*****

/* System function */

/*
    @param: const char *filename - имя файла который хотим создать
    @return : void
    @result: Создаем файл с заданным именем
*/
void MakeFile(const char *filename);

/*
    @param: const char *filename - имя файла который хотим удалить
    @return : void
    @result: Удаляем файл с заданным именем
*/
void DeleteFile(const char *filename);

/*****

```

```

/*****

/* Data preprocessing */
/*
    @param: const char *filename
    @param: struct _Grid *Grid - Структура сетки для инициализации
    @return: void
    @result: Валидация и загрузка данных из файла. При не удаче вы:
*/
void LoadData(const char *filename, struct _Grid *Grid);

/*****/

/* Debug functions */
/*
    @param: const struct _Grid *Grid
    @return: void
    @result: -
*/
void PrintGrid(const struct _Grid *Grid);

/*****/

int main()
{
    struct _Grid Grid;
    LoadData(InputFile, &Grid);           // Загрузили данные
    MakeFile(InterProcessFileForCommunication1); // Создали 1-ый файл
    MakeFile(InterProcessFileForCommunication2); // Создали 2-ой файл

    double integ_res = IntegrateTrapetcoid(&Grid);
    printf("res = %lf", integ_res);

    DeleteFile(InterProcessFileForCommunication1);
    DeleteFile(InterProcessFileForCommunication2);

    return 0;
}

double IntegrateTrapetcoid(const struct _Grid *Grid)
{
    CalcFunction_fx(Grid->A);
    double f0 = fx.f;
    CalcFunction_fx(Grid->B);
    double fn = fx.f;
    double res = (f0+fn)/2.0;

    for(int32_t i = 1; i < Grid->K; i++)
    {
        double x = Grid->A + (double)(i)*Grid->step;
        CalcFunction_fx(x);
        res += fx.f;
    }
}

```

```
    }

    return res*Grid->step;
}

void CalcFunction_fx(double x)
{
    /* Первый - расчет exp(x) */
    if ((_pid1 = fork()) == 0)
    {
        // открываем файл на запись
        _fd[0] = fopen(InterProcessFileForCommunication1, "w");

        if (_fd[0] == NULL)
        {
            fprintf(stderr, "can not open file : %s ", InterProcessFileForCommunication1);
            exit(FILE_OPEN_ERROR);
        }
        // printf("Child 1: %d Parent = %d\n", getpid(), getppid());

        uint32_t pid = getpid();
        double res_f = _exp(x);
        fprintf(_fd[0], "%d %lf", pid, res_f);
        fclose(_fd[0]);
        exit(OK);
    }

    /* Второй - расчет exp(-x) */
    else if (_pid1 > 0 && (_pid2 = fork()) == 0)
    {
        // открываем файл на запись
        _fd[1] = fopen(InterProcessFileForCommunication2, "w");
        if (_fd[1] == NULL)
        {
            fprintf(stderr, "can not open file : %s ", InterProcessFileForCommunication2);
            exit(FILE_OPEN_ERROR);
        }
        // printf("Child 2: %d Parent = %d\n", getpid(), getppid());
        uint32_t pid = getpid();
        double res_f = _exp(-x);
        fprintf(_fd[1], "%d %lf", pid, res_f);
        fclose(_fd[1]);
        exit(OK);
    }
    else
    {
        /* Родитель */
        int status1 = 0;
        int status2 = 0;
        waitpid(_pid1, &status1, NULL); // Ждем первого и его код
        waitpid(_pid2, &status2, NULL); // Ждем второго и его код

        if (status1 == OK)
```



```

11 (status1 == OK)
{
    _fd[0] = fopen(InterProcessFileForCommunication1, "r");
    if (_fd[0] == NULL)
    {
        fprintf(stderr, "Can not open file: %s", InterProcessFileForCommunication1);
        exit(FILE_OPEN_ERROR);
    }

    fscanf(_fd[0], "%d %lf", &fx.pid1, &fx.f1);
    fclose(_fd[0]);
}
else
{
    fprintf(stderr, "Error function calc status = %d", status1);
    exit(FUNCTION_CALCULATION_ERROR);
}

if (status2 == OK)
{
    _fd[1] = fopen(InterProcessFileForCommunication2, "r");
    // Аналогично верхнему
    if (_fd[1] == NULL)
    {
        fprintf(stderr, "Can not open file: %s", InterProcessFileForCommunication2);
        exit(FILE_OPEN_ERROR);
    }

    fscanf(_fd[1], "%d %lf", &fx.pid2, &fx.f2);
    fclose(_fd[1]);
}
else
{
    fprintf(stderr, "Error function calc status = %d", status2);
    exit(FUNCTION_CALCULATION_ERROR);
}

/* Объединение результата */
fx.f = (fx.f1 - fx.f2)/2.0;
}

}

void LoadData(const char *filename, struct _Grid *Grid)
{
    FILE *fd = fopen(filename, "r");

    if (fd == NULL)
    {
        fprintf(stderr, "Can not open file: %s", filename);
        exit(FILE_OPEN_ERROR);
    }

    if (fscanf(fd, "%lf %lf %d", &Grid->A, &Grid->B, &Grid->K) != 3)
    {

```

```
        fprintf(stderr, "Error read file: %s", filename);
        exit(FILE_READ_ERROR);
    }

    /* Валидация полученных данных */
    if ((Grid->A > Grid->B) || Grid->K <= 0)
    {
        fprintf(stderr, "Error input data from file: %s\n", filename);
        exit(INCORRECT_GRID_INPUT_DATA);
    }

    /* Данные корректные рассчитываем шаг */
    Grid->step = (Grid->B - Grid->A) / (double)(Grid->K);

    close(fd);
}

void MakeFile(const char *filename)
{
    char *cmd = (char *)calloc(strlen(filename) + strlen(_touch) +
    /* create cmd */
    memmove(cmd, _touch, strlen(_touch));
    strcat(cmd, filename);
    strcat(cmd, _2_dev_null);
    /* execute */
    system(cmd);

    free(cmd);
}

void DeleteFile(const char *filename)
{
    char *cmd = (char *)calloc(strlen(filename) + strlen(_rm) + stl

    /* create cmd */
    memmove(cmd, _rm, strlen(_rm));
    strcat(cmd, filename);
    strcat(cmd, _2_dev_null);
    /* execute */
    system(cmd);

    free(cmd);
}

double _exp(double x)
{
    int n = 1; // Счетчик

    double e = 1.0; // результат расчета
    double ei = 1.0; // i-ая итерация
```

```
double tmpx = -1.0 * fabs(x); // -1*|x| - для расчета функции e^-x
/* Расчет для функции e^-x */
for (int n = 1; n < 1e4; n++)
{
    ei = (ei * tmpx) / (double)n;
    e += ei;
    if (fabs(ei) < eps)
        break;
}

/* Возврат значения с учетом знака */
if (isinf(e) || isnan(e))
{
    fprintf(stderr, "double overflow or incorrect math operation\n");
    exit(DOUBLE_OVERFLOW);
}
if (x > eps)
    return 1.0 / e;
else
    return e;
}

void PrintGrid(const struct _Grid *Grid)
{
    printf("Grid data\n");
    printf("A = %lf B = %lf Step = %lf K = %u", Grid->A, Grid->B, Grid->Step, Grid->K);
}
}
```

makefile

```
# Makefile for lab #2

all: main

main: main.o
    gcc -std=c11 main.o -o main

main.o: main.c
    gcc -std=c11 -c main.c -lm -O2

clean:
    rm -rf *.o main
```

