Министерство науки и высшего образования Российской Федерации

Новосибирский государственный технический университет

Управление ресурсами в вычислительных системах

Лабораторная работа №4

Факультет: прикладной математики и информатики

Группа: ПМ-63

Студенты: Кожекин М.В.

Утюганов Д.С.

Преподаватели: Стасышин В. М.

Хайленко Е. А.

Новосибирск

2019

1. Условие (Вариант №8)

Разработать программу, вычисляющую значение f(x) как сумму ряда от k=1 до k=N от выражения (-1)^(k+1)\*x^(2k-1)/(2k-1)! для значений x, равномерно распределённых на интервале [0;Pi], и выводящую полученный результат f(x) в файл в двоичном формате. В это время предварительно подготовленный процесс-потомок читает данные из файла, преобразовывает их в текстовую форму и выводит на экран до тех пор, пока процесс-предок не передаст ему через файл ключевое слово (например, "STOP"), свидетельствующее об окончании процессов.

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.

Модифицировать ранее разработанную программу с учётом следующих требований:

* действия процесса-потомка реализуются отдельной программой, запускаемой по одному из системных вызовов execl(), execv() и т.д. из процесса-потомка;
* процесс-потомок, после порождения, должен начинать и завершать свое функционирование по сигналу, посылаемому процессом-предком (это же относится и к нескольким процессам-потомкам);
* обмен данными между процессами необходимо осуществить через программный канал.

1. Анализ задачи

**Часть 1**

Вычисляем значение ряда Тейлора на отрезке от 0 до π и сохранили результат в файл.

Создали процесс-потомок, затем считали результаты из файла и вывели их на экран.

**Часть 2**

Вычисляем значение ряда Тейлора на отрезке от 0 до π, запускаем вторую программу, создаём именованный канал, открываем этот канал для записи и сохраняем результат в него.

Вторая программа (**outputRes**) создаёт тот же именованный канал, но для чтения, и выводим результат из этого канала на экран.

1. Используемые программные средства

**Функции:**

int fork() – порождение процесса-потомка

int strtol(const char \*str, char \*\*strend, int base) – преобразование строки **str** с вещественным числом к самому числу с основанием **base**.

int execv(char \*filename, char \*argv[]) – порождение процесса-потомка и передача ему текущих параметров

floar sin(float x) – библиотечная функция вычисления синуса

mkfifo(const char \*filename, int mode) – создание именованного канала

int fd open(const char\*path, int mode) – открытие файла или именованного канала

close(const char\*file) – закрытие файла или именнованного канала.

unlink(const char\*file) – удаление файла

1. Спецификация

Программа находится в папке ~/lab4

Чтобы собрать первую часть программы нужно ввести “make lab41”, а чтобы собрать вторую “make lab42”

Чтобы запустить программу, нужно использовать команду “./lab41” или же “./lab42”

В результате работы программы, будет показано значение функции написанной нами и библиотечной функции sin(x) из <math.h> на отрезке от 0 до π.

1. Исходный код

**Часть 1**

Файл lab41.c

|  |
| --- |
| #include <stdlib.h>  #include <stdio.h>  #include <math.h>  const float pi = 3.14159265358979323846;  /\*\*  @function f  @brief вычисляет sin(x) через ряд Тейлора (-1)^(k+1) \* x^(2k-1) / (2k-1)!  @param x - число, число членов ряда  @return вещественный элемент  \*/  float f(float x, int N) {  float result = 0.0;  float tmp = 0.0;  for (int k = 1; k < N; ++k) {  tmp = pow(x, (2 \* k - 1));  for (int i = 1; i <= 2 \* k - 1; ++i)  tmp /= i;  if ((k + 1) % 2 == 0)  result += tmp;  else  result -= tmp;  }  return result;  }    int main(int argc, char \*argv[])  {  if (argc > 1 && strcmp(argv[1], "--help") == 0) {  printf("Инструкция:\n");  printf("Запуск программы происходит командой ./lab41 K, N, где K - число членов ряда Тейлора, а N - количество интервалов от 0 до PI\n");  printf("Если введено неверное число параметров (не равно 2), то выводится сообщение об ошибке.\n");  return 0;  }  else if (argc > 3) {  printf("Ошибка: слишком длинный список параметров. Должно быть два параметра.\n");  return -1;  }  else if (argc < 3) {  printf("Ошибка: слишком короткий список параметров. Должно быть два параметра.\n");  return -1;  }  char \*end;  // K - число членов ряда Тейлора, а N - количество интервалов от 0 до PI  int K = strtol(argv[1], &end, 10), N = strtol(argv[2], &end, 10);  // Вычисляем sin(x), x = 0,PI  // и сохраняем в бинарный временный файл  FILE \*tmpW = fopen("tmp\_file", "wb");  for (int i = 0; i <= N; ++i) {  float x = pi \* i / N;  fprintf(tmpW, "%d\t%f\n", i, f(x, K));  }  fclose(tmpW);  if (fork() == 0) { // Процесс-потомок  FILE \*tmpR = fopen("tmp\_file", "rb");  float array\_f[N + 1]; // Результаты вычислений  float x, calculatedValue;  int index;  for (int i = 0; i <= N; ++i) {  fscanf(tmpR, "%d\t%f\n", &index, &calculatedValue);  array\_f[index] = calculatedValue;  }  printf("\ni\tx\t\tf(x)\t\tsin(x)\n");  for (int i = 0; i <= N; ++i) {  x = pi \* i / N;  printf("%d\t%f\t%f\t%f\n", i, x, array\_f[i], sin(x));  }  fclose(tmpR);  return 0;  }  return 0;  } |

**Часть 2**

lab42.c

|  |
| --- |
| #include <sys/types.h>  #include <sys/stat.h>  #include <unistd.h>  #include <fcntl.h>  const float pi = 3.14159265358979323846;  /\*\*  @function f  @brief вычисляет sin(x) через ряд Тейлора (-1)^(k+1) \* x^(2k-1) / (2k-1)!  @param x - число, число членов ряда  @return вещественный элемент  \*/  float f(float x, int N) {  float result = 0.0;  float tmp = 0.0;  for (int k = 1; k < N; ++k) {  tmp = pow(x, (2 \* k - 1));  for (int i = 1; i <= 2 \* k - 1; ++i)  tmp /= i;  if ((k + 1) % 2 == 0)  result += tmp;  else  result -= tmp;  }  return result;  }    int main(int argc, char \*argv[])  {  if (argc > 1 && strcmp(argv[1], "--help") == 0) {  printf("Инструкция:\n");  printf("Запуск программы происходит командой ./lab41 K, N, где K - число членов ряда Тейлора, а N - количество интервалов от 0 до PI\n");  printf("Если введено неверное число параметров (не равно 2), то выводится сообщение об ошибке.\n");  return 0;  }  else if (argc > 3) {  printf("Ошибка: слишком длинный список параметров. Должно быть два параметра.\n");  return -1;  }  else if (argc < 3) {  printf("Ошибка: слишком короткий список параметров. Должно быть два параметра.\n");  return -1;  }  char \*end;  // K - число членов ряда Тейлора, а N - количество интервалов от 0 до PI  int K = strtol(argv[1], &end, 10), N = strtol(argv[2], &end, 10);  // Массив вычисленных значений синуса  float array\_f[N + 1];  // Вычисляем sin(x), x = 0,PI  for (int i = 0; i <= N; ++i) {  float x = pi \* i / N;  array\_f[i] = f(x,K);  }  if (fork() == 0) {  // Порождение процесса-потомка  if (execv("outputRes", argv) == -1){  printf("Ошибка создания процесса потомка.\n");  return -1;  }  }  // Создаём именованный канал  char \* myfifo = "/tmp/myfifo";  mkfifo(myfifo, O\_WRONLY);  int fd = open(myfifo, O\_WRONLY);  // и передаём по каналу дочернему процессу  for (int i = 0; i <= N; ++i)  write(fd, &array\_f[i], sizeof(array\_f[i]));  close(fd);  unlink(myfifo);  return 0;  } |

outputRes.c

|  |
| --- |
| #include <stdio.h>  #include <sys/types.h>  #include <sys/stat.h>  #include <fcntl.h>  const float pi = 3.14159265358979323846;  int main(int argc, char \*argv[]) {  char \*end;  int K = strtol(argv[1], &end, 10), N = strtol(argv[2], &end, 10);  // Открываем именованный канал  char \* myfifo = "/tmp/myfifo";  int fd = open(myfifo, O\_RDONLY);  float number, x;  printf("\ni\tx\t\tf(x)\t\tsin(x)\n");  // Получаем значения от процесса-предка по каналу  for (int i = 0; i <= N; ++i) {  read(fd, &number, sizeof(float));  x = pi \* i / N;  printf("%d\t%f\t%f\t%f\n", i, x, number, sin(x));  }  close(fd);  return 0;  } |

**Содержимое файла Makefile**

|  |
| --- |
| # Makefile for lab4  lab41:  gcc lab41.c -o lab41 -lm  lab42:  gcc lab42.c -o lab42 -lm  gcc outputRes.c -o outputRes -lm  clean:  rm lab41 lab42 outputRes |

1. Результат работы программы

Как и ожидалось обе версии программы выдают один и тот же результат

|  |
| --- |
| -bash-4.2$ ./lab41 10 10  -bash-4.2$  i x f(x) sin(x)  0 0.000000 0.000000 0.000000  1 0.314159 0.309017 0.309017  2 0.628319 0.587785 0.587785  3 0.942478 0.809017 0.809017  4 1.256637 0.951057 0.951057  5 1.570796 1.000000 1.000000  6 1.884956 0.951056 0.951057  7 2.199115 0.809017 0.809017  8 2.513274 0.587785 0.587785  9 2.827433 0.309017 0.309017  10 3.141593 -0.000000 -0.000000 |
| -bash-4.2$ ./lab42 10 10  i x f(x) sin(x)  -bash-4.2$ 0 0.000000 0.000000 0.000000  1 0.314159 0.309017 0.309017  2 0.628319 0.587785 0.587785  3 0.942478 0.809017 0.809017  4 1.256637 0.951057 0.951057  5 1.570796 1.000000 1.000000  6 1.884956 0.951056 0.951057  7 2.199115 0.809017 0.809017  8 2.513274 0.587785 0.587785  9 2.827433 0.309017 0.309017  10 3.141593 -0.000000 -0.000000 |