Министерство образования и науки Российской Федерации
Новосибирский государственный технический университет

Лабораторная работа № 4

Порождение нового процесса и работа с ним. запуск программы в рамках порожденного процесса. Сигналы и каналы в ОС UNIX

Факультет	ПМИ
Группа	ПМ-01
Бригада	2
Студенты	Александров М.Е.
	Жигалов П.С.
Преподаватели	Быханов К.В.
	Саутин А.С.

Новосибирск
2013

Постановка задачи

1. Разработать программу, вычисляющую значение f(x) как сумму ряда от k=0 до k=N от выражения 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. Должны быть учтены следующие требования:

- 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. 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.

Описание метода решения задачи

Часть 1.

После проверки на корректность исходных данных, задачу будем решать в соответствии со следующим алгоритмом:

1) Создать временный двоичный файл для взаимодействия процессов. Файл должен содержать следующее число-флаг: 0 - родительский процесс может внести новую порцию информации, 1 - в файле находятся значения для процесса-потомка, -1 - окончание работы. В случае, когда число 1, за ним следуют еще два числа: значение x и значение f(x).

2) Создать процесс-потомок, ожидающий данных от процесса-предка и выводящий их как только они появились.

3) Процесс-предок помещает в файл новую порцию данных.

4) Когда все данные обработаны, записать в файл -1 и завершить работу.

Часть 2

После проверки на корректность исходных данных, задачу будем решать в соответствии со следующим алгоритмом:

1) Создать неименованный канал для взаимодействия процессов.

2) Создать процесс-потомок, ожидающий соответствующего сигнала начала работы.

3) Процесс-предок должен записать в неименованный канал порции из двух чисел (значение x и значение f(x)) и послать порожденному процессу сигнал начала работы.

4) Процесс-потомок должен вывести полученные данные.

5) Процесс-предок посылает сигнал о завершении работы и завершается, как только завершится потомок.

Описание программного средства

Часть 1.

Программа написана на языке C++. Для получения исполняемого файла исходный текст про-граммы следует скомпилировать каким-либо компилятором языка C++. Исполняемый файл может ра-ботать под любой UNIX-совместимой ОС, поддерживающей соответствующий формат исполняемого файла.

Запуск программы осуществляется командой:

./summator <numsteps>

где <numsteps> – число отрезков, на которые разбивается интервал [0;Pi]. Вывод производится на экран (stdout). Если программе передается неверное число параметров или неверное значение – программа выдаст сообщение об этом. Также программе требуется файл ./config.txt, в котором находится число N.

Часть 2.

Программа написана на языке C++. Для получения исполняемого файла исходный текст про-граммы следует скомпилировать каким-либо компилятором языка C++. Исполняемый файл может ра-ботать под любой UNIX-совместимой ОС, поддерживающей соответствующий формат исполняемого файла.

Запуск программы осуществляется командой:

./summator_parent <numsteps>

где <numsteps> – число отрезков, на которые разбивается интервал [0;Pi]. Вывод производится на экран (stdout). Если программе передается неверное число параметров или неверное значение – программа выдаст сообщение об этом. Также программе требуется файл ./config.txt, в котором находится число N.

Исходный текст

Часть 1

Файл summator.cpp

```
/**
 * Program for compute value of f (x) as sum of series from k = 0 to k = N
 * on expression x ^ (2k +1) / (2k +1)! for values of x, uniformly distributed
 * on interval [0; Pi].
 *
 * @author      Alexandrov Mikhail
 * @author      Zhigalov Peter
 * @version     0.3
 */
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/wait.h>
```

```
#include <math.h>

/**
 * Function for calculate sum of series
 *
 * @param x argument of function
 * @param N number elements on series
 * @return sum of series
 */
double f(double x, int N)
{
    double sum = x;
    int tmp;
    for (int i = 1; i < N; i++)
    {
        tmp = i * 2;
        sum += sum * x * x / tmp / (tmp + 1);
    }
    return sum;
}

/**
 * Main function
 *
 * @param argc number of program arguments
 * @param argv program arguments
 * @return     1 - incorrect parameters, 0 - all correct
 */
int main(int argc, char* argv[])
{
    /** Check arguments number */
    if (argc != 2)
    {
        fprintf(stderr, "1 arguments expected.\n");
        exit(1);
    }
    /** Config file loading */
    FILE *file = fopen("./config.txt", "r");
    if (!file)
    {
        fprintf(stderr, "Can't open configuration file.\n");
        exit(1);
    }
    int N = -1;
    if (!fscanf(file, "%i", &N) || N == -1)
    {
        fprintf(stderr, "Can't read configuration file.\n");
        fclose(file);
        exit(1);
    }
    fclose(file);
    /** Arguments loading */
    int num_steps = 1;
    if (!sscanf(argv[1], "%i", &num_steps))
    {
        fprintf(stderr, "Can't read arguments.\n");
        exit(1);
    }
    if (num_steps <= 0)
    {
        fprintf(stderr, "Bad arguments.\n");
        exit(1);
    }
    double h = M_PI / num_steps;

    pid_t child_pid;
    struct flock lock, lock_info;
    double data[2];
    int flag = 0, temp;

    const char *temp_file = "/tmp/summator_msg"; /**< Temp file */
    int fd;
    fd = open(temp_file, O_RDWR | O_TRUNC | O_CREAT, 0644);
    if (fd == -1)
```

```
{
    fprintf(stderr, "Can't open temp file.\n");
    exit(1);
}
close(fd);

switch(child_pid = fork())
{
/** Can`t create fork */
case (pid_t)-1:
    fprintf(stderr, "Error creatinfg fork.\n");
    break;

/** Code for forked process */
case (pid_t)0:
{
    child_pid = getpid();
    /** Some problems with temp file */
    if ((fd = open(temp_file, O_RDWR)) == -1)
    {
        fprintf(stderr, "Can't open temp file.\n");
        exit(1);
    }
    /** While not STOP flag */
    while (flag != -1)
    {
        lock.l_len = 0;
        lock.l_start = 0;
        lock.l_whence = SEEK_SET;
        lock.l_type = F_WRLCK;
        /** Waiting unlocking and do lock */
        while (fcntl(fd, F_SETLK, &lock) == -1)
        {
            fcntl(fd, F_GETLK, &lock_info);
        }
        lseek(fd, 0, SEEK_SET);
        read(fd, &flag, sizeof(int));
        /** Print result */
        if (flag == 1)
        {
            read(fd, data, 2 * sizeof(double));
            printf("f(%lf) = %lf\n", data[0], data[1]);
            lseek(fd, 0 ,SEEK_SET);
            temp = 0;
            write(fd, &temp, sizeof(int));
        }
        lock.l_len = 0;
        lock.l_start = 0;
        lock.l_whence = SEEK_SET;
        lock.l_type = F_UNLCK;
        /** Unlocking file */
        fcntl(fd, F_SETLK, &lock);
        usleep(10000);
    }
    break;
}

/** Code for parent process */
default:
{
    int fd = open(temp_file, O_RDWR);

    for(int i = 0; i <= num_steps; i++)
    {
        double x = h * i;
        usleep(10000);
        lock.l_len = 0;
        lock.l_start = 0;
        lock.l_whence = SEEK_SET;
        lock.l_type = F_WRLCK;
        /** Waiting unlocking and do lock */
        while (fcntl(fd, F_SETLK, &lock) == -1)
        {
            fcntl(fd, F_GETLK, &lock_info);
```

```
            }
            lseek(fd, 0, SEEK_SET);
            read(fd, &flag, sizeof(int));
            if (flag == 0)
            {
                lseek(fd, 0, SEEK_SET);
                temp = 1;
                write(fd, &temp, sizeof(int));
                data[0] = x;
                data[1] = f(x, N);
                write(fd, data, 2 * sizeof(double));
            }
            lock.l_len = 0;
            lock.l_start = 0;
            lock.l_whence = SEEK_SET;
            lock.l_type = F_UNLCK;
            /** Unlocking file */
            fcntl(fd, F_SETLK, &lock);
        }

        /** Ending work */
        usleep(10000);
        lock.l_len = 0;
        lock.l_start = 0;
        lock.l_whence = SEEK_SET;
        lock.l_type = F_WRLCK;
        /** Waiting unlocking and do lock */
        while (fcntl(fd, F_SETLK, &lock) == -1)
        {
            fcntl(fd, F_GETLK, &lock_info);
        }
        lseek(fd, 0, SEEK_SET);
        read(fd, &flag, sizeof(int));
        /** Write STOP flag */
        if (flag == 0)
        {
            lseek(fd, 0, SEEK_SET);
            temp = -1;
            write(fd, &temp, sizeof(int));
        }
        lock.l_len = 0;
        lock.l_start = 0;
        lock.l_whence = SEEK_SET;
        lock.l_type = F_UNLCK;
        /** Unlocking file */
        fcntl(fd, F_SETLK, &lock);
        wait(&temp);
        break;
    }
    }
    close(fd);
    return 0;
}
```

Часть 2

Файл summator_parent.cpp

```
/**
 * Program for compute value of f (x) as sum of series from k = 0 to k = N
 * on expression x ^ (2k +1) / (2k +1)! for values of x, uniformly distributed
 * on interval [0; Pi].
 * It is a parent program.
 *
 * @author      Alexandrov Mikhail
 * @author      Zhigalov Peter
 * @version     0.3
 */
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/wait.h>
#include <math.h>
```

```
/** Signal about begin of work */
#define SIGNAL_BEGIN SIGUSR1
/** Signal about end of work */
#define SIGNAL_END SIGUSR2

/**
 * Function for exit with code 2 if child unexpectedly changed status
 */
void child_dead(int)
{
    fprintf(stderr, "Child unexpectedly changed status.\n");
    exit(2);
}

/**
 * Function for calculate sum of series
 *
 * @param x     argument of function
 * @param N     number elements on series
 * @return sum  of series
 */
double f(double x, int N)
{
    double sum = x;
    int tmp;
    for (int i = 1; i < N; i++)
    {
        tmp = i * 2;
        sum += sum * x * x / tmp / (tmp + 1);
    }
    return sum;
}

/**
 * Main function
 *
 * @param argc  number of program arguments
 * @param argv  program arguments
 * @return      1 - incorrect parameters, 2 - child unexpectedly changed status, 0 - all correct
 */
int main(int argc, char* argv[])
{
    /** Check arguments number */
    if (argc != 2)
    {
        fprintf(stderr, "1 arguments expected.\n");
        exit(1);
    }
    /** Config file loading */
    FILE *file = fopen("./config.txt", "r");
    if (!file)
    {
        fprintf(stderr, "Can't open configuration file.\n");
        exit(1);
    }
    int N = -1;
    if (!fscanf(file, "%i", &N) || N == -1)
    {
        fprintf(stderr, "Can't read configuration file.\n");
        fclose(file);
        exit(1);
    }
    fclose(file);
    /** Arguments loading */
    int num_steps = 1;
    if (!sscanf(argv[1], "%i", &num_steps))
    {
        fprintf(stderr, "Can't read arguments.\n");
        exit(1);
    }
    if (num_steps <= 0)
    {
        fprintf(stderr, "Bad arguments.\n");
```

```
        exit(1);
    }
    double h = M_PI / num_steps;

    pid_t child_pid;
    double data[2];

    /** Create pipe */
    int fifo[2];
    if (pipe(fifo) == -1)
    {
        fprintf(stderr, "Pipe create failed.\n");
        exit(1);
    }

    switch(child_pid = fork())
    {
        /** Can`t create fork */
    case (pid_t)-1:
        fprintf(stderr, "Error creatinfg fork.\n");
        break;

        /** Code for forked process */
    case (pid_t)0:
    {
        /** Replace stdin with pipe */
        dup2(fifo[0], 0);
        /** Execute child process */
        execv("./summator_children", NULL);
        fprintf(stderr, "Child exec failed\n");
        exit(1);
    }
    }

    /** If child changed status run child_dead function */
    signal(SIGCHLD, child_dead);

    usleep(10000);

    /** Calc */
    for(int i = 0; i <= num_steps; i++)
    {
        double x = h * i;
        data[0] = x;
        data[1] = f(x, N);
        write(fifo[1], data, 2 * sizeof(double));
    }

    /** Send begin signal */
    kill(child_pid, SIGNAL_BEGIN);

    /** Some wait */
    usleep(10000);
    /** Ignoring if child changed status */
    signal(SIGCHLD, SIG_IGN);

    /** Send end signal */
    kill(child_pid, SIGNAL_END);

    /** Some wait */
    int t;
    wait(&t);

    /** Close pipe */
    close(fifo[1]);
    close(fifo[0]);
    return t;
}
```

Файл summator_children.cpp

```
/**
 * Program for compute value of f (x) as sum of series from k = 0 to k = N
 * on expression x ^ (2k +1) / (2k +1)! for values of x, uniformly distributed
 * on interval [0; Pi].
```

```
 * It is a child program.
 *
 * @author      Alexandrov Mikhail
 * @author      Zhigalov Peter
 * @version     0.3
 */
#include <stdlib.h>
#include <stdio.h>
#include <signal.h>
#include <unistd.h>

/** Signal about begin of work */
#define SIGNAL_BEGIN SIGUSR1
/** Signal about end of work */
#define SIGNAL_END SIGUSR2

/**
 * Function for exit with code 0 if end signal received
 */
void end(int)
{
    exit(0);
}

/**
 * Function for beginning print if begin signal received
 */
void begin(int)
{
    double data[2];

    while(read(0, data, 2 * sizeof(double)) > 0)
        printf("f(%lf) = %lf\n", data[0], data[1]);

    /** If received end signal run end function */
    signal(SIGNAL_END, end);
    /** Wait end signal */
    sigpause(SIGNAL_END);
}

/**
 * Main function
 *
 * @return 1 if error, 0 - all correct
 */
int main()
{
    /** If received begin signal run begin function */
    signal(SIGNAL_BEGIN, begin);
    /** Wait begin signal */
    sigpause(SIGNAL_BEGIN);
    return 1;
}
```

Пример выполнения

Часть 1

Тест № 1

Запуск: ./summator

Содержимое файла с параметрами "config.txt": 10

Вывод на экран:

```
1 arguments expected.
```

Тест № 2

Запуск: ./summator 42 42

Содержимое файла с параметрами "config.txt": 10

Вывод на экран:

```
1 arguments expected.
```

Тест № 3

Запуск: ./summator -100

Содержимое файла с параметрами "config.txt": 10

Вывод на экран:

```
Bad arguments.
```

Тест № 4

Запуск: ./summator 10

Содержимое файла с параметрами "config.txt": файл отсутствует.

Вывод на экран:

```
Can't open configuration file.
```

Тест № 5

Запуск: ./summator 10

Содержимое файла с параметрами "config.txt": файл пуст.

Вывод на экран:

```
Can't read configuration file.
```

Тест № 6

Запуск: ./summator 1

Содержимое файла с параметрами "config.txt": 3

Вывод на экран:

```
f(0.000000) = 0.000000
f(3.141593) = 12.409783
```

Тест № 7

Запуск: ./summator 10

Содержимое файла с параметрами "config.txt": 1000

Вывод на экран:

```
f(0.000000) = 0.000000
f(0.314159) = 0.323762
f(0.628319) = 0.707508
f(0.942478) = 1.223589
f(1.256637) = 1.971771
f(1.570796) = 3.100237
f(1.884956) = 4.837139
f(2.199115) = 7.539675
f(2.513274) = 11.771574
f(2.827433) = 18.426286
f(3.141593) = 28.923486
```

Часть 2

Тест № 8

Запуск: ./summator_parent

Содержимое файла с параметрами "config.txt": 10

Вывод на экран:

```
1 arguments expected.
```

Тест № 9

Запуск: ./summator_parent 42 42

Содержимое файла с параметрами "config.txt": 10

Вывод на экран:

```
1 arguments expected.
```

Тест № 10

Запуск: ./summator_parent -100

Содержимое файла с параметрами "config.txt": 10

Вывод на экран:

```
Bad arguments.
```

Тест № 11

Запуск: ./summator_parent 10

Содержимое файла с параметрами "config.txt": файл отсутствует.

Вывод на экран:

```
Can't open configuration file.
```

Тест № 12

Запуск: ./summator_parent 10

Содержимое файла с параметрами "config.txt": файл пуст.

Вывод на экран:

```
Can't read configuration file.
```

Тест № 13

Запуск: ./summator_parent 1

Содержимое файла с параметрами "config.txt": 3

Вывод на экран:

```
f(0.000000) = 0.000000
f(3.141593) = 12.409783
```

Тест № 14

Запуск: ./summator_parent 10

Содержимое файла с параметрами "config.txt": 1000

Вывод на экран:

```
f(0.000000) = 0.000000
f(0.314159) = 0.323762
f(0.628319) = 0.707508
f(0.942478) = 1.223589
f(1.256637) = 1.971771
f(1.570796) = 3.100237
f(1.884956) = 4.837139
f(2.199115) = 7.539675
f(2.513274) = 11.771574
f(2.827433) = 18.426286
f(3.141593) = 28.923486
```