Министерство науки и образования РФ  
Новосибирский государственный технический университет  
Кафедра ТПИ

Лабораторная работа №4

по дисциплине «Управление ресурсами в вычислительных системах»

Факультет: ПМИ

Группа: ПМИ-31

Студенты: Суслов А.В.

Эльвейн Н.Л.  
 Олимпиади М.Б.

Преподаватель: Хайленко Е.А.  
 Филиппова Е.В.

Вариант: 1

1. **Цель работы**

Изучить программные средства создания процессов, получить навыки

управления и синхронизации процессов, а также простейшие способы обмена

данными между процессами. Ознакомиться со средствами динамического запуска

программ в рамках порожденного процесса, изучить механизм сигналов ОС UNIX,

позволяющий процессам реагировать на различные события, и каналы, как одно

из средств обмена информацией между процессами.

1. **Задание**

I. Разработать программу, реализующую действия, указанные в задании к

лабораторной работе с учетом следующих требований:

1) все действия, относящиеся как к родительскому процессу, так и к

порожденным процессам, выполняются в рамках одного исполняемого файла;

2) обмен данными между процессом-отцом и процессом-потомком

предлагается выполнить посредством временного файла;

3) 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.

II. Модифицировать ранее разработанную программу с учетом следующих

требований:

1) действия процесса-потомка реализуются отдельной программой,

запускаемой по одному из системных вызовов execl(), execv() и т.д. из процесса-

потомка;

2) процесс-потомок, после порождения, должен начинать и завершать свое

функционирование по сигналу, посылаемому процессом-предком (это же

относится и к нескольким процессам-потомкам);

3) обмен данными между процессами необходимо осуществить через

программный канал.

Вариант №1: Разработать программу, вычисляющую интеграл на отрезке [A;B] от функции exp(x) методом трапеций, разбивая интервал на K равных отрезков. Для нахождения exp(х) программа должна породить процесс, вычисляющий её значение путём разложения в ряд по формулам вычислительной математики.

1. **Описание разработанного программного средства**

Программа написана на языке C++. Для получения исполняемого файла исходный текст программы следует скомпилировать каким-либо компилятором языка C++. Исполняемый файл может работать под любой UNIX-совместимой ОС, поддерживающей соответствующий формат исполняемого файла и систему procfs.

Запуск программы осуществляется командой:

./lab4\_rm\_self [2> dev/null]

Поддерживаемые аргументы командной строки:  
 без аргументов - интерактивный режим  
 **-x <value>** вычисляет значение экспоненты в точке value и завершает работу  
 **-i** устанавливает "улучшенный режим" из части II задания  
 **-q <string>** задаёт входные параметры, они не будут запрашиваться более  
 **-s** устанавливает простой режим, вычисляет результат, не используя fork, pipe, exec

1. **Текст программы**

#include <stdio.h>

#include <getopt.h>

#include <signal.h>

#include <unistd.h>

#include <stdlib.h>

#include <sys/wait.h>

#include <errno.h>

#include <string.h>

#include "common.h"

#pragma region "Global vars"

#define FMT\_W "%.16lf"

#define RETURN\_FAILURE -0.123456789

#define BUFSIZE 512

#define RD 0 // Read end of pipe

#define WR 1 // Write end of pipe

int pid = 0;

char \*path;

FILE \*logfile;

bool improved\_mode = false;

#pragma endregion

double exponent(double x) { // this function is identical to exp(x). thanks to N. Elveyn

double ans = 1, power = 1;

double abs = 1, fac = 1;

for (int i = 1; abs > 1e-6; i++) {

fac \*= i;

power \*= x;

abs = power / fac;

ans += abs;

}

return ans;

}

void send\_thru\_file(double value) {

FILE \*f = fopen("/tmp/lab4.out", "w");

fprintf(f, FMT\_W"\n", value);

fflush(f);

fclose(f);

return;

}

void send\_thru\_pipe(double value) {

return; // this is done automatically via stdout redirection

}

void recv\_thru\_file(pid\_t pid, double &x) {

FILE \*f = fopen("/tmp/lab4.out", "r");

if(f == NULL || fscanf(f, "%lf", &x) < 1) {

fprintf(logfile, "[!] File is empty or does not exists\n");

x = RETURN\_FAILURE;

}

return;

}

void recv\_thru\_pipe(pid\_t pid, double &x) {

ssize\_t rd; char inbuf[BUFSIZE];

if(((rd = read((int)pid, inbuf, BUFSIZE-1)) <= 0)||(sscanf(inbuf, "%lf", &x) < 1))

x = RETURN\_FAILURE;

return;

}

void(\*send\_response)(double x) = send\_thru\_file;

void(\*recv\_response)(pid\_t pid, double &x) = recv\_thru\_file;

double exp\_fork(double x) {

// piping by http://stackoverflow.com/a/26384903/1543625

int ctoppipe[2]; // Child-to-parent pipe

pid\_t childpid; // Process ID of child

int wdup; // dup():ed stdout for child

pipe(ctoppipe);

// signals by http://citforum.ru/programming/unix/signals/

sigset\_t newset;

sigemptyset(&newset);

sigaddset(&newset, SIGCONT);

sigprocmask(SIG\_BLOCK, &newset, 0);

switch(childpid = fork()) {

case -1: printf("[!] Could not fork\n"); break;

case 0: {

int pp = getpid();

// write logfile

char args[BUFSIZE] = "\0";

sprintf(args, "/tmp/lab4-%d.log", pp);

FILE \*f = fopen(args, "w");

setbuf(f, NULL);

fprintf(f, "Child process PID = %d got x = "FMT\_W"\n", pp, x);

getcwd(args, BUFSIZE - 1);

fprintf(f, "pwd = %s\n", args);

double ret = 0;

if (improved\_mode) {

close(ctoppipe[RD]);

int sig;

while(sigwait(&newset, &sig) < 0 && sig != SIGCONT);

if(sig == SIGCONT) fprintf(stderr, "SIGCONT received\n");

else fprintf(stderr, "Signal %d received\n", sig);

int wdup = dup2(ctoppipe[WR], STDOUT\_FILENO);

sprintf(args, "-x"FMT\_W, x);

// thanks to Hemul1995

pp = execl(path, path, "-i", args, 0);

if(pp != 0) // return code from child

fprintf(f,"[!] Child process [-i] '%s' '%s' exited with code %d\n", path, args, pp);

}

else {

ret = exponent(x);

send\_response(ret);

}

fclose(f);

exit(0);

}

break;

}

// parent only

close(ctoppipe[WR]);

double ret = 0;

int a = 0;

if(improved\_mode)

kill(childpid, SIGCONT);

int b = wait(&a); // for inline

if (a != 0) {

printf("[!] Child process exited with code %d, halt\n", a);

exit(3);

}

recv\_response((pid\_t)ctoppipe[RD], ret);

fprintf(stderr, "Read '"FMT\_W"' from file/stream\n", ret);

return ret;

}

int main(int argc, char \*\*argv) {

path = new char[BUFSIZE];

// open logfile

pid = getpid();

sprintf(path, "/tmp/lab4-%d-main.log", pid);

logfile = fopen(path, "w");

setbuf(logfile, NULL);

// get itself

memset(path, 0, BUFSIZE);

if(readlink("/proc/self/exe", path, BUFSIZE - 1) < 0) { // could not find itself

printf("[x] I could not find myself :(\n");

exit(4);

}

fprintf(logfile, "Process PID = %d, PPID = %d started\n", pid, getppid());

fprintf(logfile, "Command line: ");

for(int i = 1; i < argc; i++)

fprintf(logfile, "%s ", argv[i]);

fprintf(logfile, "\nExecutable itself: %s\n", path);

// mathematics here

double a = 0, b = 1, S = 0;

uint K = 1;

int e = 0;

double(\*func)(double x) = &exp\_fork;

// command line parameters parse

bool input\_cli = false;

while ((e = getopt(argc, argv, "ix:sq:")) != -1) {

fprintf(logfile, "Got argument: %c = %s\n", e, optarg);

switch (e) {

case 'q': {

if (sscanf(optarg, "%lf %lf %u", &a, &b, &K) < 3)

printf("Not enough input data; using interactive mode.");

else input\_cli = true;

}

break;

case 'i': {

send\_response = &send\_thru\_pipe;

recv\_response = &recv\_thru\_pipe;

improved\_mode = true;

}

break;

case 'x': {

sscanf(optarg, "%lf", &a);

fprintf(logfile, "Embedded calc got "FMT\_W"\n", a);

fprintf(stdout, FMT\_W"\n%c", exponent(a), 0);

fclose(logfile);

exit(0);

}

break;

case 's': func = &exponent; break;

case '?': fprintf(logfile, "[!] Unknown command-line arg\n"); exit(1); break;

}

};

if (!input\_cli) {

fprintf(stdout, "Enter A, B, K:\n");

fscanf(stdin, "%lf %lf %u", &a, &b, &K);

}

// Kotes formula: S = h ((f0 + fn) / 2 + sum(1, n-1, [fi])) [+ En(f), optional]

if (K < 1)

K = 1;

double h = (b - a) / K;

double x\_curr = a;

for (int i = 1; i <= K - 1; i++) {

x\_curr += h;

S += func(x\_curr);

}

S += (func(a) + func(b)) / 2;

S \*= h;

fprintf(stdout, "Result: S ~= %f\n", S);

return 0;

}

1. **Результат работы программы**

|  |  |  |  |
| --- | --- | --- | --- |
| № | Входные данные | Выходные данные | Комментарий |
| 1 | ./lab4\_rm\_self -q "0 3 27" -i | Result: S ~= 19.105168 | Проверка работоспособности программы |
| 2 | ./lab4\_rm\_self -q "-2 -1 27" -i | Result: S ~= -0.500000 | Проверка работоспособности программы при отрицательных значениях |
| 3 | ./lab4\_rm\_self -q "0 0 1000" -i | Result: S ~= 0.000000 | Проверка работоспособности программы на большом количестве итераций. |