МИНИСТЕРСТВО НАУКИ
И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное
образовательное учреждение высшего образования
«Новосибирский государственный технический университет»

НГТУ НЭТИ | **Факультет прикладной математики и информатики**

Кафедра теоретической и прикладной информатики

Лабораторная работа №2
по дисциплине «Управление ресурсами в вычислительных системах»

Порождение нового процесса и работа с ним. Запуск программы в рамках порожденного процесса

Бригада 9	БЕГИЧЕВ АЛЕКСАНДР
Группа ПМ-92	ШИШКИН НИКИТА
Вариант 9	
Преподаватели	СИВАК МАРИЯ АЛЕКСЕЕВНА
	СТАСЫШИН ВЛАДИМИР МИХАЙЛОВИЧ

Новосибирск, 2022

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 одного из средств обмена информацией между процессами.

Вариант: Разработать программу, вычисляющую значение функции $f(x) = \pi \cdot \sinh(x)$ в точке x. Для нахождения π и $\sinh(x)$ программа должна породить два параллельных процесса, вычисляющих эти величины путем разложения в ряд по формулам вычислительной математики.

Программа на языке СИ

Описание алгоритма

```
1:  x ← value;
2:  info_proc fst, snd;                          ▷ Structure that contains PID and the value
3:  info_proc info_all[2];                       ▷ Entries that the parent reads from the file
4:  open the file;
5:  fst.pid = fork();
6:  if (fst.pid == 0) then
7:      fst.value ← calculate_pi();                                  ▷ Wallis product
8:      fst.pid = getpid();
9:      process writes to the file;
10:     exit(0);
11: end if
12: snd.pid = fork();
13: if (snd.pid == 0) then
14:     snd.value ← calculate_sinh(x);                               ▷ Taylor series
15:     snd.pid = getpid();
16:     process writes to the file;
17:     exit(0);
18: end if
19: proc_count ← 0;
20: while (proc_count != 2) do
21:     proc_count ← 0;
22:     while (have the ability to read) do      ▷ The parent is constantly reading the file
23:         proc_count++;
24:     end while
25: end while
26: info_all[2] ← file entries;
27: close the file;
28: result ← info_all[0].value · info_all[1].value;                  ▷ π · sinh (x)
```

Используемые системные функции

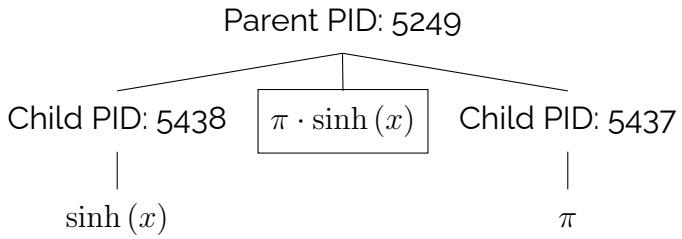
- `fork()` – вызов порождает новый процесс (процесс-потомок).
- `gepid()` – вызов предназначен для получения собственного идентификатора процесса.
- `geppid()` – вызов предназначен для получения идентификатора процесса-отца.
- `exit()` – вызов прекращает функционирование процесса.

Тестирование

Первый тест

```
[hukutka@hukutka lab_2]$ ./main
Enter x: 5
I am child with pid 5437, my parent is 5429 and I am calculating pi
I am child with pid 5438, my parent is 5429 and I am calculating sinh(x)
Child with pid 5437 calculated pi = 3.141591
Child with pid 5438 calculated sinh(x) = 74.203211
I am parent with pid 5429 and I calculated sinh(x) * pi
sinh(x) * pi = 233.116145
[hukutka@hukutka lab_2]$
```

Точное значение: $\pi \cdot \sinh(5) \approx 233.11626$



Второй тест

Создан файл без прав доступа чтения и записи, название которого совпадает с именем временного файла.

```
[hukutka@hukutka lab_2]$ ./main
Enter x: 10
Error occured while opening file
: Permission denied
sinh(x) * pi = 0.000000
[hukutka@hukutka lab_2]$
```

```
[hukutka@hukutka lab_2]$ touch data.dat
[hukutka@hukutka lab_2]$ chmod -rw data.dat
```

Листинг

```c
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h> // for fork
#include <sys/types.h> // for pid_t

#define PI_COMPUTION_MAX_ITERS 1e+6
#define SINH_ACCURACY 1e-14
#define SINH_COMPUTION_MAX_ITERS 1e+3
#define FILENAME "data.dat"


// Struct to store info
typedef struct {
    pid_t pid;
    double value;
} info_proc;


// Wallis product
double calculate_pi() {
    double result = 1.;
    double a = 2.;
    double b = 1.;

    for (int i = 0; i < PI_COMPUTION_MAX_ITERS; i++) {
        result *= a / b;

        if (a < b) {
            a += 2.;
        } else {
            b += 2.;
        }
    }

    return result * 2.;
}


// get sinh(x)
double calculate_sinh(double x) {
    int i = 1;
    double cur = x;
    double acc = 1;
    double fact = 1.;
    double pow = x;

    while (fabs(acc) > SINH_ACCURACY && i < SINH_COMPUTION_MAX_ITERS) {
        fact *= ((2. * i) * (2. * i + 1.));
        pow *= x * x;
        acc = pow / fact;
        cur += acc;
        i++;
    }

    return cur;
}
```

```c

// compute sinh(x) * PI
int compute(double x, double *result) {
    info_proc fst, snd, info_all[2];
    pid_t fst_tmp, snd_tmp;
    FILE *fp;

    if ((fp = fopen(FILENAME, "w+")) == NULL) {
        perror("Error occured while opening file\n");
        return EXIT_FAILURE;
    }

    // Create process to compute PI
    // ------------------------------------------------
    if ((fst.pid = fork()) == -1) {
        perror("Fork failed\n");
        return EXIT_FAILURE;
    }

    if (fst.pid == 0) {
        printf("I am child with pid %d, my parent is %d and I am calculating pi\n",
                fst.pid = getpid(), getppid());

        fst.value = calculate_pi();

        if (fwrite(&fst, sizeof(info_proc), 1, fp) == 0) {
            perror("Error writing file\n");
            return EXIT_FAILURE;
        }

        exit(0);
    }

    // Create process to compute sinh(x)
    // ------------------------------------------------
    if ((snd.pid = fork()) == -1) {
        perror("Fork failed\n");
        return EXIT_FAILURE;
    }

    if (snd.pid == 0) {
        printf("I am child with pid %d, my parent is %d and I am calculating
        ↪  sinh(x)\n",
                snd.pid = getpid(), getppid());

        snd.value = calculate_sinh(x);

        if (fwrite(&snd, sizeof(info_proc), 1, fp) == 0) {
            perror("Error writing file\n");
            return EXIT_FAILURE;
        }

        exit(0);
    }

    // Read file untill there will be
    // two info_proc structs
    // ------------------------------------------------
    int proc_count = 0;
```

```c
    while (proc_count != 2) {
        proc_count = 0;

        while (fread(info_all + proc_count, sizeof(info_proc), 1, fp)) {
            proc_count++;
        }

        fseek(fp, 0, SEEK_SET);
    }

    fclose(fp);

    // Remove 'data.dat'
    // ---------------------------------------------
    if (remove(FILENAME) != 0) {
        perror("Error: unable to delete the file\n");
        return EXIT_FAILURE;
    }

    // Create pretty console output
    // ---------------------------------------------
    if (info_all[0].pid == fst.pid && info_all[1].pid == snd.pid) {
        printf("Child with pid %d calculated pi = %lf\n", info_all[0].pid,
        ↪  info_all[0].value);
        printf("Child with pid %d calculated sinh(x) = %lf\n", info_all[0].pid,
        ↪  info_all[1].value);

    } else if (info_all[1].pid == fst.pid && info_all[0].pid == snd.pid) {
        printf("Child with pid %d calculated pi = %lf\n", info_all[1].pid,
        ↪  info_all[1].value);
        printf("Child with pid %d calculated sinh(x) = %lf\n", info_all[0].pid,
        ↪  info_all[0].value);

    // Or error message
    } else {
        perror("Error: Processes PIDs do not match\n");
        return EXIT_FAILURE;
    }

    *(result) = info_all[0].value * info_all[1].value;
    // Don't forget about parent process
    printf("I am parent with pid %d and I calculated sinh(x) * pi\n", getpid());
}

int main(int argc, char *argv[]) {
    double x, result;

    printf("Enter x: ");
    scanf("%lf", &x);

    compute(x, &result);

    printf("sinh(x) * pi = %lf\n", result);

    return EXIT_SUCCESS;
}
```

Вывод

В ходе выполнения лабораторной работы мы изучили программные средства создания процессов, получили навыки управления и синхронизации процессов, а также простейшие способы обмена данными между процессами. Ознакомились со средствами динамического запуска программ в рамках порожденного процесса, изучить механизм сигналов ОС UNIX, позволяющий процессам реагировать на различные события, и каналы, как одного из средств обмена информацией между процессами.