Московский Авиационный Институт

(Национальный Исследовательский Университет)

Институт №8 “Компьютерные науки и прикладная математика”

Кафедра №806 “Вычислительная математика и программирование”

Лабораторная работа №1 по курсу

«Операционные системы»

Группа: М8О-211БВ-24

Студент: Ергизов Алексей Радикович

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 01.10.25

Москва, 2025

Постановка задачи

Вариант 5.

Пользователь вводит команды вида: «число». Далее это число передается от родительского процесса в дочерний. Дочерний процесс производит проверку на простоту. Если число составное, то это число записывается в файл. Если число отрицательное или простое, то тогда дочерний и родительский процессы завершаются.

Общий метод и алгоритм решения

Использованные системные вызовы:

- pid_t fork(void); – создает дочерний процесс.
- int pipe(int *fd); – создает канал для межпроцессорного взаимодействия.
- ssize_t read(int fd, void *buf, size_t count) – возвращает количество прочитанных символов.
- int dup2(int oldfd, int newfd) – возвращает новый файловый дескриптор.
- int execl(const char *path, const char *arg, …) – запускает новую программу
- void _exit(int status) – завершает процесс.
- int usleep(useconds_t usec) – приостановка выполнения.
- pid_t waitpid(pid_t pid, int *wstatus, int options) – ожидание изменения дочернего процесса.
- pid_t wait(int *wstatus) – ожидание завершения дочернего процесса.
- int open(const char *pathname, int flags, mode_t mode) – открывает файл

Я реализовал межпроцессорное взаимодействие с помощью системных вызовов. Есть родительский процесс, который порождает дочерний процесс, проверяющий передаваемое число на простоту. Общаются между собой процессы с помощью канала, созданным функцией pipe. Пользователь общается только с родительским процессом.

Код программы

parent.c

```
#include "include/Custom.h"


int main()

{

   char buffer[101];

   ssize_t bytes_read;

   int number;

   int pipefd[2];

   pid_t pid;
```

```
StatusCode result;

result = CustomPipe(pipefd);
if (result != SUCCESS)
{
    log_error(result, "pipe creation");
    return 1;
}

result = CustomFork(&pid);
if (result != SUCCESS)
{
    log_error(result, "fork");
    CustomClose(pipefd[0]);
    CustomClose(pipefd[1]);
    return 1;
}

if (pid == 0)
{
    CustomClose(pipefd[1]);
    dup2(pipefd[0], STDIN_FILENO);
    CustomClose(pipefd[0]);

    execl("./Child", "Child", NULL);
    CustomWrite(2, "Exec failed\n", 12);
    _exit(1);
}
else
{
```

```
CustomClose(pipefd[0]);

while (1)
{
    CustomWrite(1, "Enter number: ", 14);

    bytes_read = read(0, buffer, sizeof(buffer) - 1);
    if (bytes_read <= 0)
        break;

    int len = 0;
    while (len < bytes_read && buffer[len] != '\n' && buffer[len] != '\0')
    {
        len++;
    }
    buffer[len] = '\0';

    if (len == 0)
        continue;

    number = 0;
    int i = 0;
    int is_negative = 0;

    if (buffer[0] == '-')
    {
        is_negative = 1;
        i = 1;
    }
```

```
for (; i < len && buffer[i] >= '0' && buffer[i] <= '9'; i++)
{
    number = number * 10 + (buffer[i] - '0');
}

if (is_negative)
{
    number = -number;
}

result = CustomWrite(pipefd[1], &number, sizeof(number));
if (result != SUCCESS)
{
    log_error(result, "write to child");
    break;
}

if (number < 0)
{
    CustomWrite(1, "Negative number, exiting\n", 25);
    break;
}

usleep(10000);

int status;
if (waitpid(pid, &status, WNOHANG) == pid)
{
    CustomWrite(1, "Child finished, exiting\n", 40);
    break;
```

```
            }
        }

        CustomClose(pipefd[1]);

        wait(NULL);
    }

    return 0;
}
```

Child.c

```
#include "include/Custom.h"

int is_prime(int n)
{
    if (n < 2)
        return 0;
    if (n == 2)
        return 1;
    if (n % 2 == 0)
        return 0;

    for (int i = 3; i * i <= n; i += 2)
    {
        if (n % i == 0)
            return 0;
    }
    return 1;
}
```

```
void int_to_string(int num, char *str)
{
    int i = 0;
    do
    {
        str[i++] = '0' + (num % 10);
        num /= 10;
    } while (num > 0);

    for (int j = 0; j < i / 2; j++)
    {
        char temp = str[j];
        str[j] = str[i - j - 1];
        str[i - j - 1] = temp;
    }

    str[i] = '\0';
}

int main()
{
    int number;
    ssize_t bytes_read;

    // Читаем числа из канала (от родительского процесса)
    while ((bytes_read = read(STDIN_FILENO, &number, sizeof(number))) > 0)
    {

        if (number < 0)
```

```
        {
            _exit(0);
        }

        if (is_prime(number))
        {
            _exit(0);
        }

        int fd;
        StatusCode result = CustomOpen(&fd, "composite.txt", O_WRONLY | O_CREAT |
O_APPEND, 0644);
        if (result == SUCCESS)
        {
            char num_str[20];
            int_to_string(number, num_str);

            CustomWrite(fd, num_str, strlen(num_str));
            CustomWrite(fd, "\n", 1);
            CustomClose(fd);
        }
    }

    return 0;
}
```

Custom.c

```
#include "../include/Custom.h"

const char *status_string(StatusCode status)
```

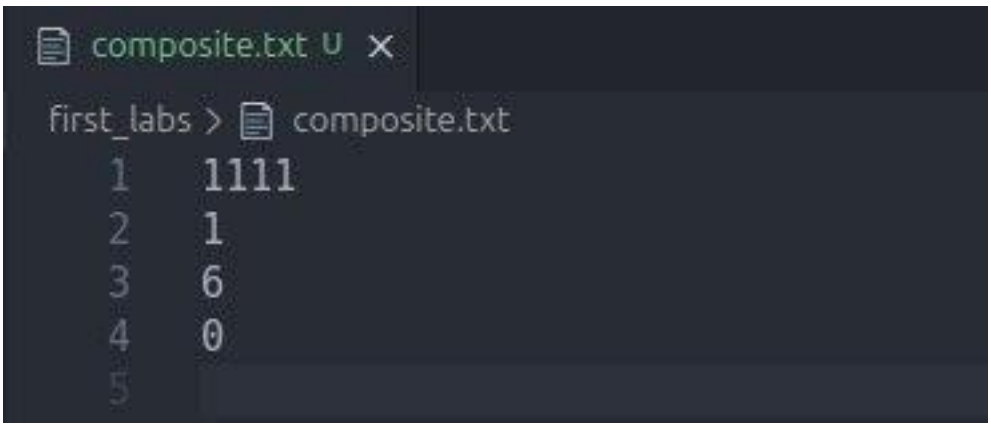
```
{
    switch (status)
    {
    case SUCCESS:
        return "success";
    case ERROR_PIPE:
        return "pipe error";
    case ERROR_FORK:
        return "fork error";
    case ERROR_WAIT:
        return "wait error";
    case ERROR_WRITE:
        return "write error";
    case ERROR_CLOSE:
        return "close error";
    case ERROR_OPEN:
        return "open error";
    case ERROR_READ:
        return "read error";
    default:
        return "unknown error";
    }
}

StatusCode CustomFork(pid_t *pid)
{
    *pid = fork();
    if (*pid == -1)
    {
        return ERROR_FORK;
```

```
    }
    return SUCCESS;
}

StatusCode CustomPipe(int *pfd)
{
    if (pipe(pfd) == -1)
    {
        return ERROR_PIPE;
    }
    return SUCCESS;
}

StatusCode CustomWait(pid_t pid)
{
    if (waitpid(pid, NULL, 0) == -1)
    {
        return ERROR_WAIT;
    }
    return SUCCESS;
}

StatusCode CustomClose(int fd)
{
    if (close(fd) == -1)
    {
        return ERROR_CLOSE;
    }
    return SUCCESS;
}
```

```
StatusCode CustomWrite(int fd, const void *buf, size_t count)
{
   if (write(fd, buf, count) == -1)
      return ERROR_WRITE;
   return SUCCESS;
}

StatusCode CustomRead(int fd, void *buf, size_t count)
{
   if (read(fd, buf, count) == -1)
      return ERROR_READ;
   return SUCCESS;
}

StatusCode CustomOpen(int *fd, const char *filename, int access, unsigned mode)
{
   *fd = open(filename, access, mode);
   if (*fd == -1)
   {
      return ERROR_OPEN;
   }
   return SUCCESS;
}

void log_error(StatusCode code, const char *context)
{
   char message[101];
   const char *err = status_string(code);
   snprintf(message, sizeof(message), "Error in %s: %s\n", context, err);
```

```
    CustomWrite(2, message, strlen(message));
}
```

Протокол работы программы



Вывод

В ходе данной лабораторной работы я научился управлять процессами в ОС. Также обеспечил обмен данных между процессами посредством каналов.