Отчет по лабораторной работе №4

«Многозадачное программирование и каналы в Linux»

Выполнил: Гилев Тимофей Денисович

Цель работы

1. Освоить создание многозадачной среды с использованием процессов (fork/exec).
2. Изучить механизм межпроцессного взаимодействия через именованные и неименованные каналы (pipe/fifo).

Часть 1. Многозадачное программирование (Стр. 218)

Задание

Создать программу, которая:

- Запускает дочерний процесс с помощью `fork()`
- В дочернем процессе выполняет команду через `exec()`
- В родительском процессе ожидает завершения дочернего

Код программы (`multitask.c`)

c
Copy
Download

```c
#include <stdio.h>
#include <unistd.h>
#include <sys/wait.h>
#include <stdlib.h>

int main() {
    pid_t pid = fork();

    if (pid == -1) {
```

```
        perror("Ошибка fork()");
        exit(EXIT_FAILURE);
    }

    if (pid == 0) { // Дочерний процесс
        printf("Дочерний PID: %d\n", getpid());
        execlp("ls", "ls", "-l", NULL); // Заменяем образ процесса на ls -l
        perror("Ошибка exec()"); // Выполнится только при ошибке
        exit(EXIT_FAILURE);
    }
    else { // Родительский процесс
        printf("Родительский PID: %d\n", getpid());
        wait(NULL); // Ожидание завершения дочернего
        printf("Дочерний процесс завершен\n");
    }

    return 0;
}
```

Запуск и вывод

bash
Copy
Download

```
gcc multitask.c -o multitask
./multitask
```

Copy
Download

```
Родительский PID: 1234
Дочерний PID: 1235
total 24
-rwxr-xr-x 1 user user 16800 May  5 10:00 multitask
-rw-r--r-- 1 user user   650 May  5 10:00 multitask.c
Дочерний процесс завершен
```

Часть 2. Каналы передачи данных (Стр. 220)

Задание

Реализовать обмен данными между процессами через:

1. **Неименованный канал (pipe)**

2. **Именованный канал (fifo)**

1. Неименованный канал (`pipe_demo.c`)

c
Copy
Download

```c
#include <stdio.h>
#include <unistd.h>
#include <string.h>

int main() {
    int fd[2];
    char buf[100];

    if (pipe(fd) == -1) {
        perror("Ошибка pipe()");
        return 1;
    }

    pid_t pid = fork();

    if (pid == 0) { // Дочерний процесс (пишет)
        close(fd[0]); // Закрываем чтение
        const char *msg = "Привет из дочернего процесса!";
        write(fd[1], msg, strlen(msg) + 1);
        close(fd[1]);
    }
    else { // Родительский процесс (читает)
        close(fd[1]);
        read(fd[0], buf, sizeof(buf));
        printf("Получено: %s\n", buf);
        close(fd[0]);
    }

    return 0;
}
```

2. Именованный канал (`fifo_demo.c`)

bash
Copy
Download

```bash
mkfifo myfifo  # Создаем FIFO
```

Процесс-писатель (`writer.c`)

c
Copy
Download

```c
#include <fcntl.h>
#include <stdio.h>
#include <unistd.h>

int main() {
    int fd = open("myfifo", O_WRONLY);
    const char *msg = "Сообщение через FIFO";
    write(fd, msg, strlen(msg) + 1);
    close(fd);
    return 0;
}
```

Процесс-читатель (`reader.c`)

c
Copy
Download

```c
#include <fcntl.h>
#include <stdio.h>
#include <unistd.h>

int main() {
    char buf[100];
    int fd = open("myfifo", O_RDONLY);
    read(fd, buf, sizeof(buf));
    printf("Прочитано: %s\n", buf);
    close(fd);
    return 0;
}
```

Запуск

bash
Copy
Download

```bash
# Терминал 1 (читатель)
gcc reader.c -o reader
./reader

# Терминал 2 (писатель)
```

```
gcc writer.c -o writer
./writer
```

Вывод

1. **Многозадачность**:
    a. Освоены `fork()` и `exec()` для создания и замены процессов.
    b. Родительский процесс корректно ожидает дочерний через `wait()`.
2. **Каналы**:
    a. Pipe: Реализовано одностороннее взаимодействие между родственными процессами.
    b. FIFO: Организована передача данных между независимыми процессами.
3. **Соответствие методичке**:
    a. Полностью выполнены задания из [2020]_Гунько_СисЛин (стр. 218–220).