Пометка к заданию №10

Задание 1:

Создается один дочерний процесс с помощью fork().

Если fork() завершился неудачно, выводится сообщение об ошибке и программа завершает работу.

Если это дочерний процесс, выводится его pid (идентификатор процесса) и ppid (идентификатор родительского процесса), затем процесс завершается.

Если это родительский процесс, выводятся его pid и ppid, после чего родитель ждет завершения дочернего процесса с помощью waitpid(). После завершения дочернего процесса, выводится его статус завершения.

```c
void task1() {
    printf("=== Задание 1 ===\n");

    pid_t pid;

    pid = fork(); // создаем новый процесс

    if (pid < 0) {
        perror("Ошибка при создании процесса");
        exit(1);
    } else if (pid == 0) {
        // Дочерний процесс
        printf("Дочерний процесс: pid = %d, ppid = %d\n", getpid(), getppid());
        exit(0);
    } else {
        // Родительский процесс
        printf("Родительский процесс: pid = %d, ppid = %d\n", getpid(), getppid());

        int status;
        waitpid(pid, &status, 0); // ждем завершения дочернего процесса
        printf("Статус завершения дочернего процесса: %d\n", WEXITSTATUS(status));
    }
}
```

Пример выполнения задания 1:

```
=== Задание 1 ===
Родительский процесс: pid = 5323, ppid = 5322
Дочерний процесс: pid = 5327, ppid = 5323
Статус завершения дочернего процесса: 0
```

***Задание 2**:*

Создается цепочка из 5 процессов, где каждый процесс (кроме последнего) создает еще один дочерний процесс.

Процессы создаются рекурсивно с помощью функции create_process, которая вызывает сама себя до достижения заданной глубины.

Каждый процесс выводит свой pid, ppid и номер процесса.

Родительский процесс ожидает завершения дочернего процесса перед завершением своей работы.

```c
void task2() {
    printf("=== Задание 2 ===\n");

    int num_processes = 5;
    int max_depth = 2;

    void create_process(int num, int depth) {
        if (num > 0) {
            pid_t pid = fork();
            if (pid < 0) {
                perror("Ошибка при создании процесса");
                exit(1);
            } else if (pid == 0) {
                // Дочерний процесс
                printf("Процесс %d: pid = %d, ppid = %d\n", num, getpid(), getppid());
                create_process(num - 1, depth - 1);
                exit(0);
            } else {
                // Родительский процесс
                wait(NULL); // ждем завершения дочернего процесса
            }
        }
    }
```

Пример выполнения задания 2:

```
Выберите задание (1, 2 или 3): 2\
=== Задание 2 ===
Процесс 5: pid = 1305, ppid = 1301
Процесс 4: pid = 1306, ppid = 1305
Процесс 3: pid = 1307, ppid = 1306
Процесс 2: pid = 1308, ppid = 1307
Процесс 1: pid = 1309, ppid = 1308
```

***Задание 3**:*

Программа ждет ввода команды от пользователя в цикле.

Если пользователь вводит exit, программа завершает работу.

Иначе создается новый дочерний процесс, который пытается выполнить введенную команду с помощью execlp().

Родительский процесс ожидает завершения дочернего и выводит код завершения команды.

```c
void task3() {
    printf("=== Задание 3 ===\n");

    char command[MAX_COMMAND_LENGTH];

    while (1) {
        printf("Введите команду (или 'exit' для выхода): ");
        fgets(command, sizeof(command), stdin);

        // Убираем символ переноса строки, если он есть
        command[strcspn(command, "\n")] = 0;

        if (strcmp(command, "exit") == 0) {
            break;
        }

        pid_t pid = fork();
        if (pid < 0) {
            perror("Ошибка при создании процесса");
            exit(1);
        } else if (pid == 0) {
            // Дочерний процесс
            execlp(command, command, NULL); // выполняем команду
            perror("Ошибка при выполнении команды");
            exit(1);
        } else {
            // Родительский процесс
            int status;
            waitpid(pid, &status, 0); // ждем завершения дочернего процесса
            printf("Команда завершилась с кодом: %d\n", WEXITSTATUS(status));
        }
```

Пример выполнения задания 3, используя команду ls:

```
Введите команду (или 'exit' ля выхода): ls
a.out  main.c
Команда завершил сь с кодом: 0
Введите команду (или 'exit' для выхода):
```