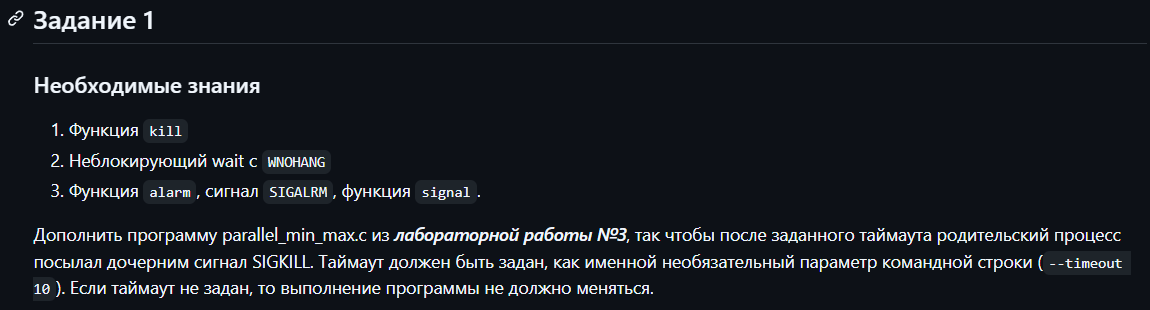
ЛР4



**1. Функция kill()**

**Назначение:**  
Отправка сигналов процессам.

* **pid**: ID процесса (>0 — конкретный процесс, 0 — всем процессам группы, -1 — всем доступным процессам).
* **sig**: Номер сигнала (например, SIGTERM, SIGKILL).

**2. Неблокирующий wait с WNOHANG**

Назначение: Проверка статуса дочерних процессов без блокировки.

**3. Функция alarm(), сигнал SIGALRM, signal()**

**alarm()**

**Назначение:**  
Установка таймера для отправки сигнала SIGALRM через указанное время.

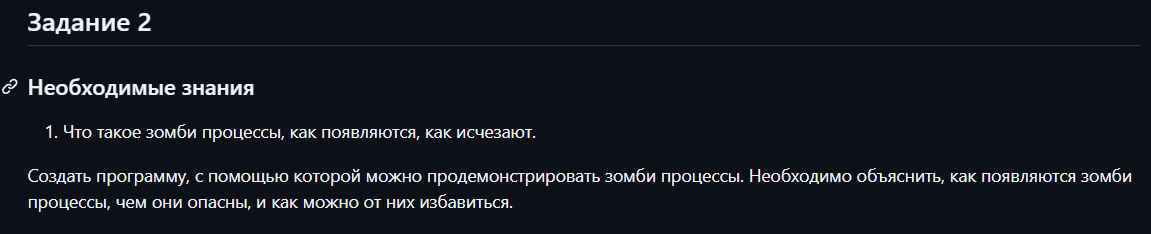
**Сигнал SIGALRM**

Стандартный сигнал для уведомления о срабатывании таймера.

**Функция signal()**

**Назначение:**  
Установка обработчика сигнала.

Программу написал



**Зомби-процессы (Zombie Processes)**

**1. Что это?**

Зомби-процесс — это **уже завершенный процесс**, который остается в таблице процессов до тех пор, пока родительский процесс не прочитает его статус завершения.

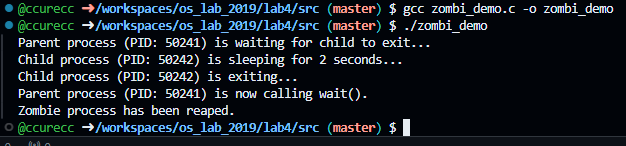
**2. Как появляются?**

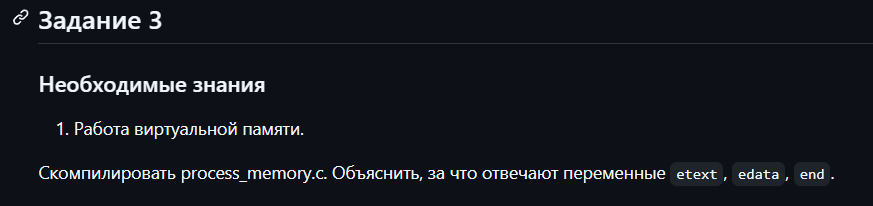
1. Дочерний процесс завершается (вызовом exit() или сигналом).
2. Родительский процесс **не вызывает** wait() или waitpid() для чтения статуса.
3. Ядро сохраняет запись о процессе (PID, код завершения), пока родитель не обработает его.

**3. Как исчезают?**

- Родитель вызывает wait()/waitpid() — зомби удаляется из таблицы процессов.

- Родитель завершается — все зомби-потомки переходят к init (PID 1), который автоматически вызывает wait().

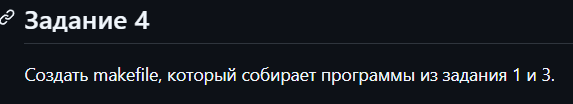




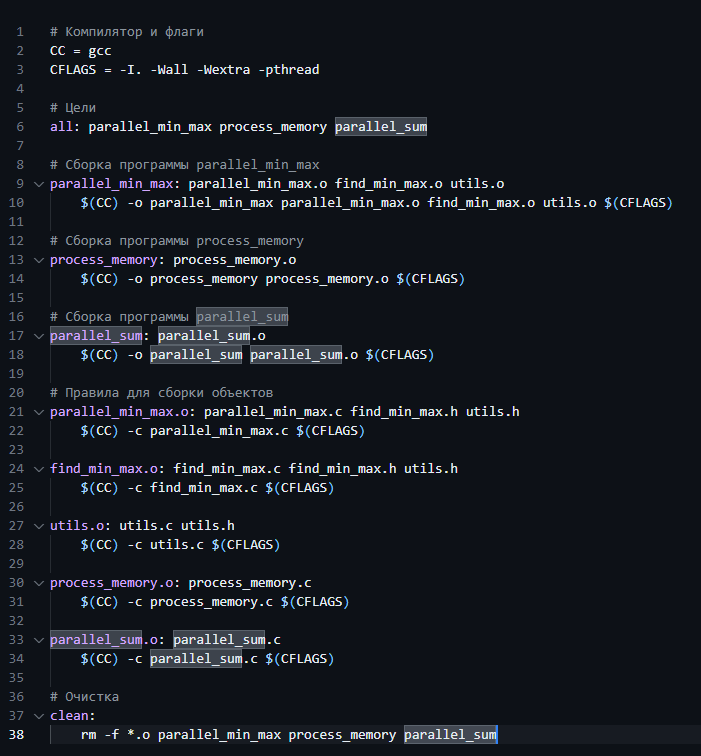
**1. Что такое виртуальная память?**

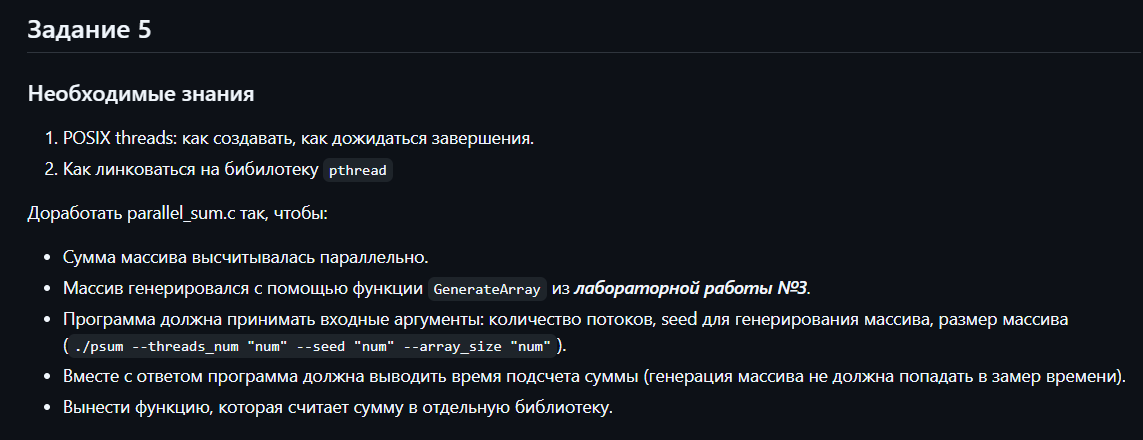
Это абстракция, предоставляемая ОС, которая позволяет программам "думать", что у них есть **непрерывное адресное пространство**, даже если физическая память фрагментирована или занята другими процессами.

1. etext - адрес конца сегмента кода (текстового сегмента). Это точка, где заканчивается исполняемый код программы.
2. edata - адрес конца сегмента инициализированных данных. Это граница между инициализированными и неинициализированными данными.
3. end - адрес конца сегмента BSS (неинициализированных данных) и начала кучи (heap).

Выполнил эти задания в одном файле





**Работа с POSIX Threads (pthreads)**

**1. Создание потока**

Используется функция pthread\_create().

#include <pthread.h>

int pthread\_create(

pthread\_t \*thread, // Указатель на идентификатор потока

const pthread\_attr\_t \*attr, // Атрибуты потока (NULL по умолчанию)

void \*(\*start\_routine)(void \*), // Функция, которую выполнит поток

void \*arg // Аргумент для функции потока

);

**2. Ожидание завершения потока**

Функция pthread\_join() блокирует выполнение, пока поток не завершится.

int pthread\_join(

pthread\_t thread, // Идентификатор потока

void \*\*retval // Указатель на возвращаемое значение (можно NULL)

);

**3. Как линковаться с библиотекой pthread**

При компиляции нужно добавить флаг -pthread (или -lpthread в старых версиях GCC).

gcc program.c –o program -pthread

Сделал