1. Что такое процесс?

\*\*Процесс\*\* — это программа в состоянии выполнения. Процесс включает в себя текущее состояние программы (контекст выполнения), такие как:

- Код программы.

- Данные программы.

- Время процессора, отведенное для выполнения.

- Используемые системные ресурсы (файлы, память и т.д.).

Каждый процесс имеет уникальный идентификатор процесса (PID) и может находиться в одном из нескольких состояний: создание, выполнение, ожидание, завершение и т.д.

2. Какие ресурсы доступны процессу?

Процессу доступны следующие ресурсы:

- \*\*Центральный процессор (CPU)\*\* — для выполнения инструкций.

- \*\*Оперативная память (RAM)\*\* — для хранения кода программы, данных, стека и динамических областей (куча).

- \*\*Дескрипторы файлов\*\* — для работы с файлами и устройствами.

- \*\*Сетевые ресурсы\*\* — для взаимодействия через сеть.

- \*\*Системное время\*\* — для выполнения операций в определенный момент.

- \*\*Регистр процессора и контекст выполнения\*\* — для отслеживания текущего состояния процесса.

- \*\*Прочие ресурсы\*\*, такие как порты ввода/вывода, устройства и т.д.

3. Что такое дочерний и родительский процесс?

- \*\*Родительский процесс\*\* — это процесс, который создает другой процесс, называемый \*\*дочерним процессом\*\*. Каждый процесс в операционной системе, за исключением начального, создается другим процессом.

- Дочерний процесс наследует часть ресурсов родительского процесса, например, копию его памяти, файловые дескрипторы и переменные среды. Однако дочерний процесс является самостоятельной сущностью с собственным PID.

4. Что такое системный вызов?

\*\*Системный вызов\*\* — это механизм, с помощью которого пользовательская программа взаимодействует с ядром операционной системы для выполнения привилегированных операций, которые напрямую недоступны программам из пользовательского пространства. Примеры таких операций: работа с файлами, создание новых процессов, управление памятью и т.д.

5. Какие системные вызовы предназначены для создания процессов в Windows и Linux?

- \*\*В Windows\*\* для создания процессов используется системный вызов `CreateProcess()`.

- \*\*В Linux\*\* системный вызов для создания процессов — это `fork()`, а также его вариации, такие как `vfork()` и `clone()`.

6. Какие функции из WinAPI и POSIX связаны с ранее названными системными вызовами?

- \*\*В Windows\*\* API-функция, связанная с системным вызовом `CreateProcess()`, — это одноимённая функция `CreateProcess()`, которая позволяет создать новый процесс и выполнить в нем программу.

- \*\*В POSIX (Linux/Unix)\*\* функции `fork()` и `vfork()` непосредственно связаны с системными вызовами того же имени и используются для создания дочернего процесса.

7. Сколько новых процессов появится в результате выполнения программой следующей последовательности вызовов fork()?

Каждый вызов `fork()` создает новый процесс. В случае трёх последовательных вызовов `fork()` количество новых процессов можно рассчитать следующим образом:

- Первый вызов `fork()` создаст один новый процесс, итого — 2 процесса (родительский + дочерний).

- Второй вызов `fork()` создаст новый процесс в каждом существующем процессе (то есть 2 новых процесса), итого — 4 процесса.

- Третий вызов `fork()` создаст еще по одному процессу в каждом из существующих (то есть 4 новых процесса), итого — 8 процессов.

Итого, программа создаст \*\*7 новых процессов\*\* (8 всего с родительским процессом).

8. Для чего нужен exec?

Функция `exec()` (или семейство функций `exec`, таких как `execl`, `execv` и т.д.) в системах Unix и Linux используется для замены текущего образа процесса новым образом программы. Это означает, что процесс продолжает существовать, но его код и данные заменяются новой программой, переданной функции `exec()`. Функция `exec()` не создает новый процесс, она только изменяет текущий.

Это полезно, когда процесс (например, после вызова `fork()`) должен запустить другую программу.