**Написание многопоточного сервера**

**Цель:** Изучить основы сетевой подсистемы, типичные причины проблем безопасности программ, закрепить знания взаимодействия процессов и потоков.

**Критерии оценивания:**

**25 - 34**- однопоточный сервер + клиент по заданному условию.  
 **35 - 50**- многопоточный сервер по заданному условию, включающему в себя модель сетевого сервиса, а также клиент.  
 **+1 - +10** лишних баллов за написание эксплоита, использующего уязвимости вашего сервера. Максимальный бонус только за эксплоит, который после своего завершения оставляет рабочий, а не засегфолтившийся после отработки эксплоита сервер.

***Разработку необходимо вести с использованием системы контроля версий Git. При сдаче задачи будет просматриваться история коммитов.***

**Срок выполнения** - до второй аттестации.

Работа выполняется по вариантам, вариант можно получить у меня. Вариант задаёт тип сервера и модель сетевого сервиса.

**Требования к различным типам серверов:**

**HTTP-сервер**  
 Необходимо реализовать GET-запрос, который отдает клиенту содержимое файла имеющего соответствующее расположение.

<http://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol>

**FTP-сервер**  
 Реализация команд LIST и CWD  
<http://en.wikipedia.org/wiki/File_Transfer_Protocol>

**SSH-сервер**  
 Речь идет о подобии ssh, т.к. никакого шифрования использоваться не будет. Обмен между сервером и клиентом осуществляется по простому сокету. Необходимо реализовать аутентификацию (имена и пароли можно хранить в текстовом файле). Должна быть возможность выполнять удаленные команды с выводом результата.

**Модели сетевых сервисов:**

**fork 1**

Данная модель создает новый процесс для каждого нового клиента. При запуске сервиса должна быть возможность указывать максимальное количество одновременно запущенных процессов. В случае, если количество клиентов превышает заданный лимит, сервер принудительно закрывает сокет. По окончанию обслуживания клиента процесс уничтожается.

**prefork 1**

Данная модель отличается от предыдущей тем, что головной процесс создает дочерние, которые будут обрабатывать запросы клиентов, заранее. Когда клиент обращается к серверу, то обслуживание передается одному из существуающих процессов. Головной процесс должен следить за тем, чтобы количество созданных процессов, ожидающих запросов клиентов, поддерживалось на заданном уровне. По окончанию обслуживания клиента процесс уничтожается.

**prefork 2**

Данная модель схожа с предыдущей, но по завершению обслуживания процессы не уничтожаются, а переходят в состояние “ожидания”. При появлении нового запроса клиента, ожидающий процесс пробуждается и выполняет все необходимые действия.

**thread1**

Аналогично fork1, но вместо процессов используются потоки.

**thread2**

Аналогично prefork1, но вместо процессов используются потоки.

**thread3**

Аналогично prefork2, но вместо процессов используются потоки.