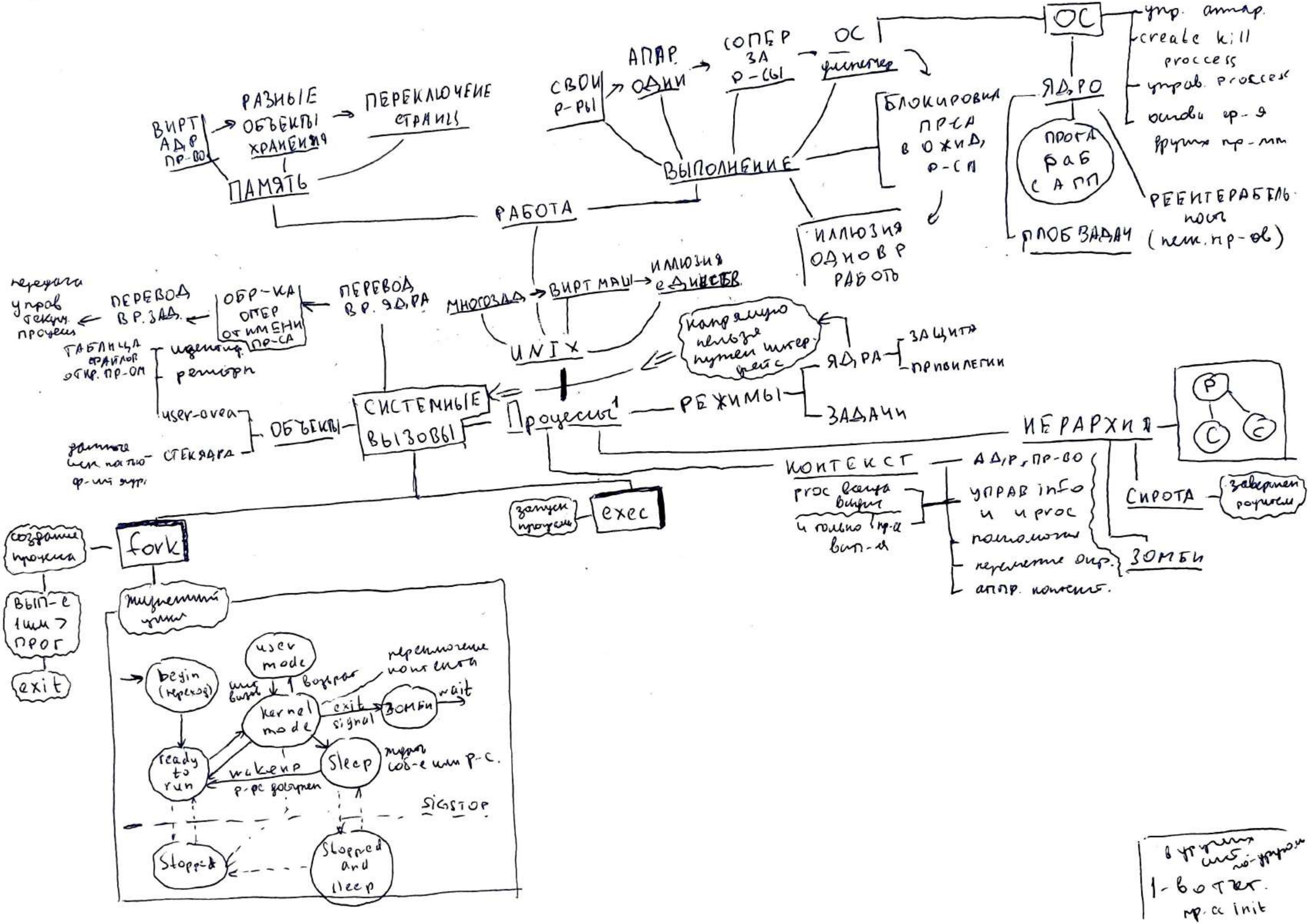
ОС
упр. аппар.
create kill process
управ. process
ЯДРО
СВОИ Р-РЫ
АППАР. ОДИН
СОПЕР ЗА Р-СЫ
ОС
БЛОКИРОВКА ПР-СА В ОЖИД. Р-СА
ВИРТ АДР ПР-ВО
РАЗНЫЕ ОБЪЕКТЫ ХРАНЕНИЯ
ПЕРЕКЛЮЧЕНИЕ СТРАНИЦ
ПАМЯТЬ
ВЫПОЛНЕНИЕ
РАБОТА
ИЛЛЮЗИЯ ОДНОВР РАБОТЫ
ПЕРЕВОД ВР. ЯДРА
ОБР-КА ОПЕР ОТ ИМЕНИ ПР-СА
ПЕРЕВОД ВР. ЗАД.
МНОГОЗАД
ВИРТ МАШ
ИЛЛЮЗИЯ
UNIX
ЯДРА
ЗАЩИТА
ПРИВИЛЕГИИ
РЕЖИМЫ
ЗАДАЧИ
СИСТЕМНЫЕ ВЫЗОВЫ
Процессы[1]
ОБЪЕКТЫ
user-area
СТЕК ЯДРА
ИЕРАРХИЯ
P
C
C
КОНТЕКСТ
АДР. ПР-ВО
УПРАВ info и proc
СИРОТА
ЗОМБИ
exec
fork
exit
user mode
kernel mode
begin
ready to run
Sleep
wakeup
exit signal
ЗОМБИ
wait
SIGSTOP
Stopped
Stopped and sleep
1 - в отчет. пр. cc init

Системный вызов fork() выполняет следующие действия:

1. Резервируется пространство свопинга для данных и стека процесса-потомка.
2. Назначается идентификатор PID и структура proc потомка.
3. Инициализируется структура proc потомка. Некоторые поля proc структуры копируются от процесса-родителя: идентификатор пользователя и группы, маски сигналов и группа процессов. Часть полей инициализируется. Остальная часть полей инициализируется специфическими для потомка значениями: PID потомка и его родителя, указатели на структуру proc родителя.
4. Создаются карты трансляции адресов для процесса-потомка.
5. Выделяется область и потомка и в нее копируется область и процесса-предка.
6. Изменяются ссылки области и на новые карты адресации и пространство свопинга.
7. Потомок добавляется в набор процессов, которые разделяют область кода программы, выполняемой процессом-родителем.
8. Посраничко дублируются области данных и стека родителя и модифицируются карты адресации потомка.
9. Потомок получает ссылки на разделяемые ресурсы, которые он наследует: открытые файлы (потомок наследует дескрипторы) и текущий рабочий каталог.
10. Инициализируется аппаратный контекст потомка путем копирования регистров родителя.
11. Помещается процесс потомок в очередь готовых процессов.
12. Возвращается PID в точку возврата из системного вызова в родительском процессе и 0 — в процессе-потомке.

Системный вызов exec выполняет следующие действия:

1. Разбирает путь к исполняемому файлу и осуществляет доступ к нему.
2. Проверяет, имеет ли вызывающий процесс полномочия на выполнение файла.
3. Читает заголовок и проверяет, что он действительно исполняемый.
4. Если для файла установлены биты SUID или SGID, то эффективные идентификаторы UID и GID вызывающего процесса изменяет на UID и GID соответствующие владельцу файла.
5. Копирует аргументы, переданные в exec, а также переменные среды в пространство ядра, после чего текущее пользовательское пространство готово к уничтожению.
6. Выделяет пространство свопинга для областей данных и стека.
7. Высвобождает старое адресное пространство и связанное с ним пространство свопинга. Если же процесс был создан при помощи vfork, происходит возврат старого адресного пространства родительскому процессу.
8. Выделяет карты трансляции адресов для нового текста, данных и стека.
9. Устанавливает новое адресное пространство. Если область текста активна (какой-то другой процесс уже выполняет эту программу), то она будет совместно использоваться с этим процессом. В других случаях пространство должно инициализироваться из выполняемого файла. Процессы в системе UNIX обычно разбиты на страницы, что означает, что каждая страница считывается в память только по мере необходимости.
10. Копирует аргументы и переменные среды обратно в новый стек приложения.
11. Сбрасывает все обработчики сигналов в действия, определенные по умолчанию, так как функции обработчиков

сигналов не существуют в новой программе. Сигналы, которые были транспортированы или заблокированы перед вызовом exec, остаются в тех же состояниях.

12. Инициализируется аппаратный контекст. При этом большинство регистров сбрасывается в 0, а указатель команд получает значение точки входа программы.