ОС. Семинар 2

21.09.24

UNIX. Процессы.

Основные концепт: ps – инфа о процессах
ls – инфа о файлах.

Состояние (name):

I. **D (U на MacOS)** uninterruptible wait
Непрерываемый сон
Например, блокировку процесса ожидания ввода/вывода прервать нельзя.

II. **S (I)** sleeping
В MacOS спит: S < 20 сек, I > 20 сек
Прерываемый сон. Ожидание прерыв. соб-я. Т.е. сон ожидающ. событие => можно прервать.

III. **R** runnable
Выполняется / может выполняться.
Системе не различает - выполняется процесс или стоит в очереди готовых на выполн-е процессов. Планировщику нужно поставить процесс в очередь г.п.в., а диспетчер выделяет процессу квант вр-ни.
Если процесс стоит в очереди, ему выделяется квант.

IV. **T** stopped terminated
Остановленный процесс.

V. **Z** zombie
Мёртвый (зомби) процесс.

Флаги (Flags) В macOS <sys.proc.h>

1	Был вызван сис.вызов fork(), но exec() не был вызван().
0	Был вызван с.в. fork(), был вызван с.в. exec().
4	SuperUser. — Мы для компьютера не существуем. ! Компьютер работает с процессами. · Терминал - процесс. · Приглашение ввода - процесс запр. ввод. (в ожидании ввода). — процесс, который имеет доступ к структур. функциям ядра.

- Любой процесс может создать любое кол-во процессов системным вызовом fork().
- В системе всегда есть процесс с pid 0 - это процесс, запустивший систему (init)
- Процесс с pid 1 - процесс, открывший терминал.
- (На macOS отсутствует процесс с pid 2) это **Kernel deamon** - системный демон, который управляет потоками.

В Unix никто не стал создать процесс, а потоки требуют мало ресурсов, приводят к проблемам при работе с памятью.

Все процессы, запущенные на данном терминале, являются потомками терминального процесса.

Процесс в Unix создаётся сис. вызовом - fork()

Команда p -ajx - показать инфу о всех процессах.

Главная особенность UNIX

Иерархия процессов поддерживается рядом указателей. Т.е., каждый процесс содержит дескриптор, который содержит:

- указатель на родительский процесс;
- указатель на родственные процессы (потомков);

! Самый молодой живущий потомок (?)

- Если завершается процесс предок, то его потомок усыновляется терминальным процессом (либо посредником между ними).

Введём системный вызов **wait()**, для того, чтобы процесс-предок не завершался до его процессов-потомков, простыми словами - чтобы не возникали сироты.

С.6. wait() – блокирует процесс-предок до завершения его процессов-потомков.

Это приводит к другой проблеме:
Возможна ситуация, когда процесс создал потомка и тот сразу завершился, а предок не успел вызвать wait(). ! Если не совершить след. действий, то процесс-предок будет блокирован навсегда.

Поэтому введено состояние зомби.

Зомби – это процесс, у которого отобраны все ресурсы, кроме последнего – дескриптора.

Дескриптор выделяется процессу при его создании.

В дескрипторе имеется поле state, которое отражает причину завершения процесса.

Когда предок дойдет до wait(), он сможет получить статус завершения потомка и продолжить выполнение.

Процесс на основе анализа (?) может ветвиться.

! Система кода не контролирует.

Соотв-но, если в коде есть сис. вызов wait() – он будет выполнен, если нет – то процесс потомок будет завершен при завершении род-предка.

fork() внутри child (?)

! Рассказать про флаг copy-on-write