Процессы

Лекция №10 по АКОС

Что такое процесс

- Процесс - это экземпляр программы в одном из состояний выполнения
- Процесс - это изолированное виртуальное адресное пространство в UNIX-системах

Аттрибуты процесса

Память:

- Значения регистров процессора
- Таблицы и каталоги страниц виртуального адресного пространства
- Private и Shared страницы памяти
- Отображение файлов в память
- Отдельный стек в ядре для обработки системных вызовов

Аттрибуты процесса

Файловая система:

- Таблица файловых дескрипторов
- Текущий каталог

  почему нет программы `cd`?
- Корневой каталог

  `root` его может менять
- Маска аттрибутов создания нового файла `umask`

Аттрибуты процесса

Другие аттрибуты:

- Переменные окружения
- Лимиты
- Счетчики ресурсов
- Идентификаторы пользователя и группы

Информация о процессах

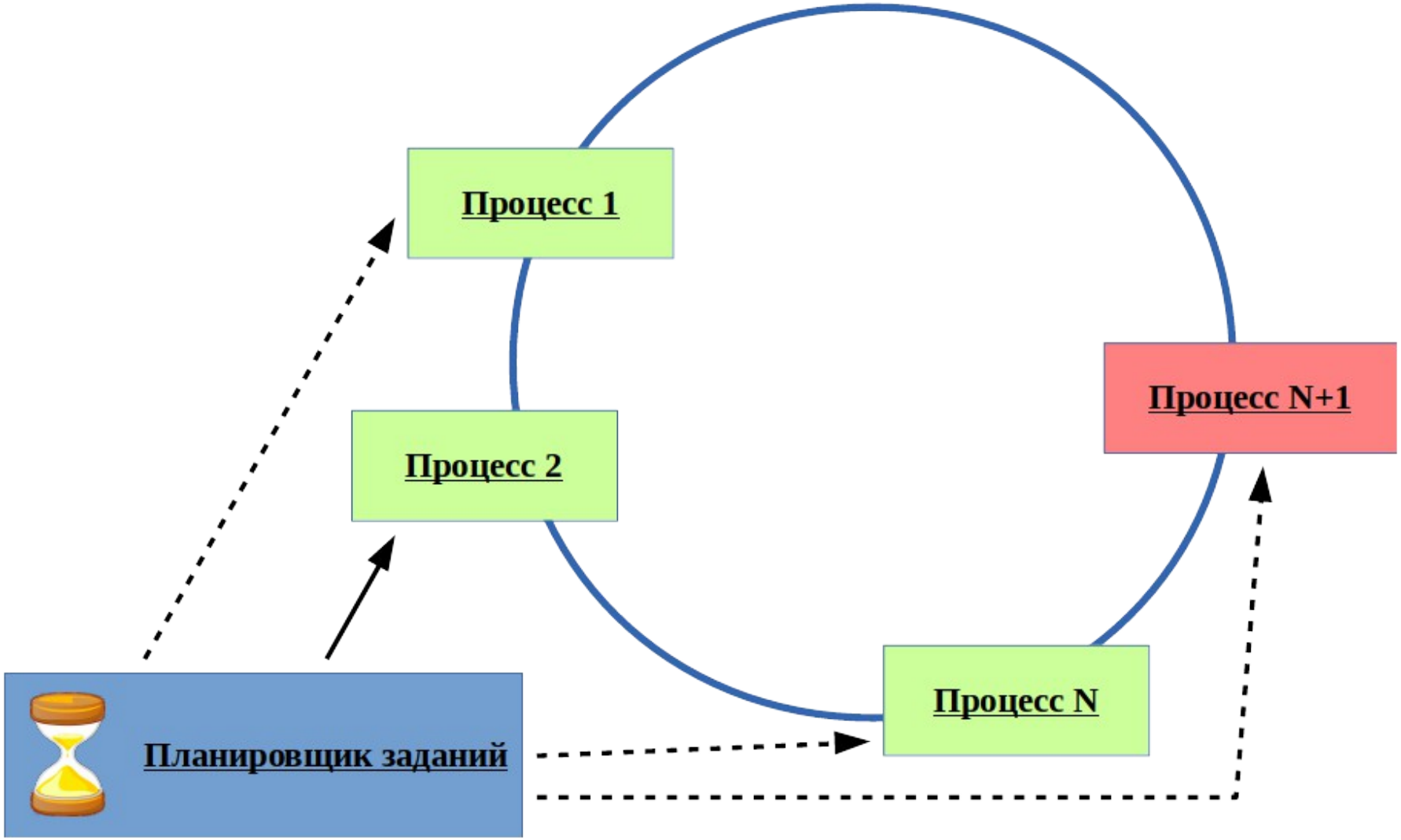
- Команда `ps` - список процессов
- Программа `top` - потребление ресурсов
- Файловая система `/proc`

Состояния процессов

- Running - выполняется
- Suspended - ожидает события
- sTopped - приостановлен
- Zombie - зомби

Round Robin

Windows 9x, старые UNIX



Приоритет

- Значение от -20 (самый высокий) до +19 (самый низкий)
- Численное значение - сколько раз пропустить планировщиком заданий
- Команды nice и renice
- Системеный вызов nice(int inc)
- Только root может повышать приоритет

Multilevel Queue

Linux, xBSD, Mac, Wantuz

sched_yield

```
while (1) {
    // do nothing - just waste CPU
}
```

```
while (1) {
    sched_yield(); // OK
}
```

Создание процесса

`pid_t resut = fork()`

Создаёт **копию** текущего процесса

- `-1 == result` -- ошибка
- `0 == result` -- для дочернего процесса
- `0 < result` -- для родительского процесса, тогда `result` - это Process ID

Пример

```
#include <sys/types>
#include <stdio.h>
#include <unistd.h>
#include <sys/wait.h>

int main() {
  pid_t  result = fork();
  if (-1==result) { perror("fork :-("); exit(1); }
  if (0==result) { printf("I'm son!\n"); }
  else {
     printf("I'm parent!\n");
     int status;
     waitpid(result, &status, 0);
     printf("Child exited with status %d\n", status);
  }
}
```

Копия процесса

- Память, регистры etc. - точная копия (кроме регистра `%eax`/`%rax`)

- НЕ копируются:
  - Process ID [`getpid()`], Parent ID [`getppid()`]
  - Сигналы, ожидающие доставки
  - Таймеры
  - Блокировки файлов

Эффекты копирования

```
int main() {
  printf("abrakadabra ");
  pid_t  result = fork();
  if (0==result) {
    printf("I'm son\n");
  }
  else {
    printf("I'm parent\n");
  }
}
```

abrakadabra I'm son
abrakadabra I'm parent

Ограничения

- `/proc/sys/kernel/pid_max` [32768] максимальное число одновременно запущенных процессов
- `/proc/sys/kernel/threads-max` [91087] максимальное число одновременно выполняющихся потоков (каждый процесс - уже один поток)

shell> **:(){ :|:& };:**

Дисклеймер!
Экспериментируйте на свой страх и риск. А если этот код скомпилировать и распространять, - то уже попадаете под действие УК РФ.

```
void fork_bomb() {
  pid_t p;
  do {
    p = fork();
  } while (-1 != p);
  while (1) sched_yeild();
}
```

Дерево процессов

- Процесс с номером `1` - ~~**`init`**~~ **`systemd`**
- У каждого процесса, кроме ~~**`init`**~~ **`systemd`** есть свой родитель
- Если родитель умирает, то его родителем становится процесс с номером `1`
- Если ребёнок умирает, про это узнаёт его родитель

Завершение работы процесса

- Системный вызов `_exit(int)`
- Функция `exit(int)`
- Оператор `return INT` в `main`

```
printf("abrakadabra");
_exit(0)
```

```
printf("abrakadabra");
exit(0)
```

Завершение работы

- Функция `exit`:
  - вызывает обработчики завершения, зарегистрированные функцией `atexit`
  - сбрасывает потоки stdio
  - удаляются файлы, созданные `tmpfile`
  - вызывается системный вызов `_exit`

Ожидание завершения процесса

- `pid_t waitpid(pid_t pid, int *status, int flags)`

- `pid` - ID процесса, или -1 для произвольного дочернего, или <1 для группы процессов
- `status` - куда записать результат работы
- `flags` - флаги ожидания:
  - 0 - по умолчанию
  - WNOHANG - не ждать, а только проверить
  - WUNTRACED - считать событие sTopped

Код возврата

- Процесс может завершиться добровольно, используя `_exit(0<=code<=255)`
- Процесс может быть принудительно завершён сигналом

```
int status;
waitpid(child, &status, 0);
if (WIFEXITED(status)) {
   printf("Exit code: %d", WEXITSTATUS(status));
}
else if (WIFSIGNALED(status)) {
   printf("Terminated by %d signal", WTERMSIG(status));
}
```

Zombie (<defunc>) - процессы

- После своего завершения процесс ещё не похоронен - его статус zombie
- Удалением зомби из таблицы процессов занимается родитель - вызовом `wait` или `waitpid`
- Если зомби не удалять - получается эффект fork-бомбы