Процессы

Лекция №11 по АКОС
2019-2020

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`

Жизненный цикл процесса

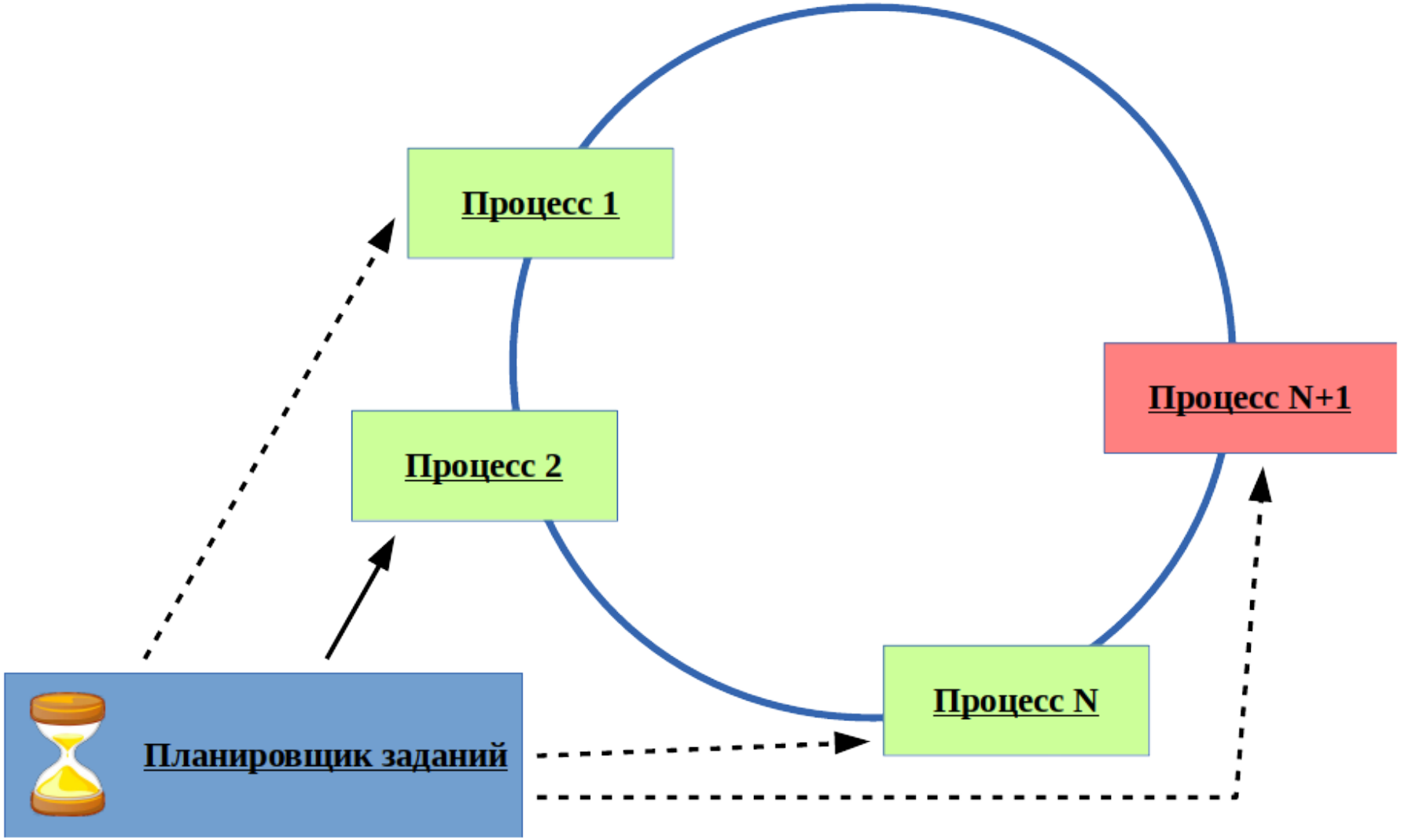
- Выполняется - ***R***unning
- Остановлен - s***T***opped
- Временно приостановлен
  - ***S***uspended - может быть завершен
  - ***D***isk Suspended - не может быть завершен
- Исследуется - ***t***racing
- Зомби - ***Z***ombie

Примеры переходов

```
sleep(10);
// переход из R в S

read(0, buffer, sizeof(buffer));
// возможен переход из R в S

read(fd, buffer, sizeof(buffer)); // где fd - файл
// возможен переход из R в D

_exit(5);
// переход из R в Z

raise(SIGSTOP);
// переход из R в T
```

Round Robin

Windows 9x, старые UNIX

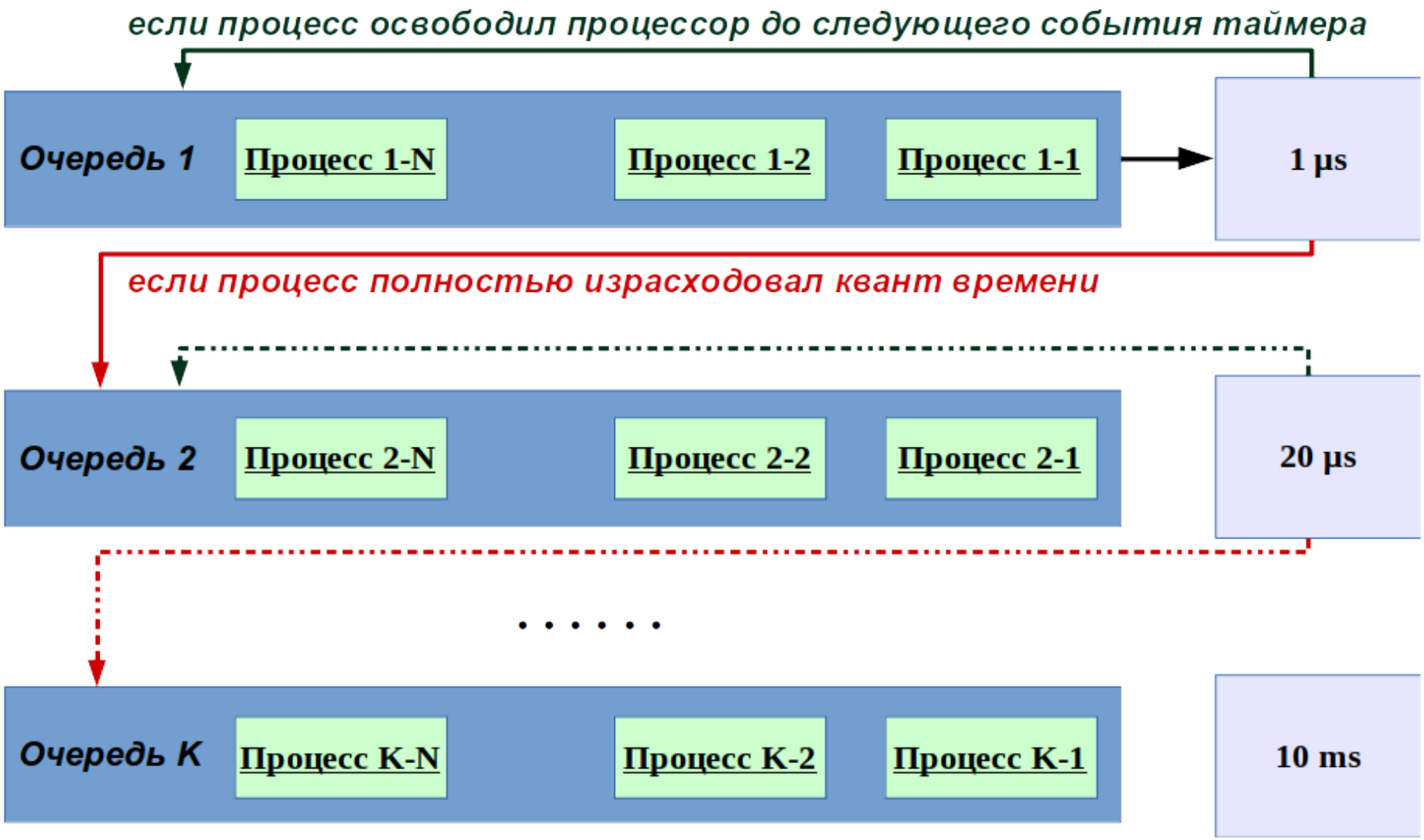


Приоритет

- 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, Windows



Ничегонеделание

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

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

Создание процесса

```
pid_t resut = fork()
```

Создаёт **<u>копию</u>** текущего процесса

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

Пример

```c
#include <sys/types.h>
#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);
  }
}
```

Демонстрация: поведение на разных ОС

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

```c
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();
}
```

Демонстрация: форк-бомба

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

(это не милый котик, а какой-то кот Шрёдингера)

exec - замещение тела процесса другой программой

man 3 exec

```
int execl(const char *path, const char *arg, ..., /* 0 */)
int execlp(const char *path, const char *arg, ..., /* 0 */)
int execle(const char *path, const char *arg, ..., /* 0 */, char * envp[])

int execv(const char *path, char * const argv[])
int execvp(const char *path, char * const argv[])

#ifdef _GNU_SOURCE
int execvpe(const char *path, char * const argv[], char * const envp[])
#endif
```

exec - замещение тела процесса другой программой

- Передача параметров
  - 'l' – переменное число аргументов
  - 'v' – массив параметров
- Передача переменных окружения
  - 'e' – дополнительно задается envp
- Поиск программы в PATH
  - имя программы может быть коротким

Признак конца строки – '\0'
Признак конца массива – NULL

exec - замещение тела процесса другой программой

```
int execvpe(const char *path, char * const argv[], char * const envp[])

int main(int argc, char *argv[])
int main(int argc, char *argv[], char *envp[]) // not portable!

char *getenv(const char *name); // POSIX
```

Команда env - отображение переменных окружения
Примеры:

- PATH - где искать программы
- LD_LIBRARY_PATH - где ld должна искать библиотеки
- HOME - домашний каталог

Пример использования

```
int main() {
  pid_t  pid = fork();
  if (-1==pid) { perror("fork :-("); exit(1); }
  if (0==pid) {
    execlp("ls", "ls", "-l", NULL);
    perror("exec :-(");
    exit(2);
  }
  else {
    waitpid(pid, NULL, 0);
  }
}
```

Windows API

```
BOOL CreateProcessA
(
    LPCTSTR lpApplicationName,                // имя исполняемого модуля
    LPTSTR lpCommandLine,                     // Командная строка
    LPSECURITY_ATTRIBUTES lpProcessAttributes, // Указатель на структуру
    LPSECURITY_ATTRIBUTES lpThreadAttributes,  // Указатель на структуру
    BOOL bInheritHandles,                     // Флаг наследования текущего процесса
    DWORD dwCreationFlags,                    // Флаги способов создания процесса
    LPVOID lpEnvironment,                     // Указатель на блок среды
    LPCTSTR lpCurrentDirectory,               // Текущий диск или каталог
    LPSTARTUPINFO lpStartupInfo,              // Указатель на структуру
    LPPROCESS_INFORMATION lpProcessInformation // Указатель на структуру
);
```

Запуск программы в Windows

```
STARTUPINFO si;
ZeroMemory(&si, sizeof(si));
PROCESS_INFORMATION pi;
BOOL success = CreateProcessA(
                  "C:\\Windows\\System32\\cmd.exe",
                  NULL,   // lpCommandLine
                  NULL,   // lpProcessAttributes
                  NULL,   // lpThreadAttributes
                  FALSE,  // bInheritHandles
                  NULL,   // dwCreationFlags
                  NULL,   // lpEnvironment
                  NULL,   // lpCurrentDirectory
                  &si,    // lpStartupInfo
                  &pi     // lpProcessInformation
                );
```

Пример использования

```
int main() {
  pid_t  pid = fork();
  if (-1==pid) { perror("fork :-("); exit(1); }
  if (0==pid) {

    // а здесь можно настроить процесс
    // до запуска программы

    execlp("ls", "ls", "-l", NULL);
    perror("exec :-(");
    exit(2);
  }
  else {
    waitpid(pid, NULL, 0);
  }
}
```

Пример использования

```
int main() {
  pid_t  pid = fork();
  if (-1==pid) { perror("fork :-("); exit(1); }
  if (0==pid) {
    chdir("/usr/bin");
    int fd = open("/tmp/out.txt",
                  O_WRONLY|O_CREAT|O_TRUNC, 0644);
    dup2(fd, 1); close(fd);
    execlp("ls", "ls", "-l", NULL);
    perror("exec :-(");
    exit(2);
  }
  else {
    waitpid(pid, NULL, 0);
  }
}
```

Демонстрация: настройка дочернего процесса

Аттрибуты процесса, сохраняемые exec

- Открытые файловые дескрипторы
- Текущий каталог
- Лимиты процесса
- UID, GID
- **Корневой каталог - только root**

SUID-флаг

- Дополнительный аттрибут выполняемого файла
- Означает, что файл запускается от имени того пользователя, который является владельцем файла

setuid / getuid v.s. geteuid

- setuid(uid_t) - установить Effective UID
- getuid() - получить реальный UID
- geteuid() - получить effective UID

Настройка параметров процесса *(только Linux)*

```
int main() {
  pid_t  pid = fork();
  if (-1==pid) { perror("fork :-("); exit(1); }
  if (0==pid) {
    // Только для PowerPC
    prctl(PR_SET_ENDIAN, PR_ENDIAN_PPC_LITTLE, 0, 0, 0);
    // Убивать все дочерние процессы при смерти родителя
    prctl(PR_SET_DEATHSIG, SIGTERM, 0, 0, 0)
    execlp("ls", "ls", "-l", NULL);
    perror("exec :-(");
    exit(2);
  }
  else {
    waitpid(pid, NULL, 0);
  }
}
```

Установка лимитов
(POSIX 2003+)

```
int main() {
  pid_t  pid = fork();
  if (-1==pid) { perror("fork :-("); exit(1); }
  if (0==pid) {
    struct rlimit lim;
    getrlimit(RLIMIT_STACK, &lim);
    lim.rlim_cur = 64 * 1024 * 1024;
    setrlimit(RLIMIT_STACK, &lim)
    execlp("ls", "ls", "-l", NULL);
    perror("exec :-(");
    exit(2);
  }
  else {
    waitpid(pid, NULL, 0);
  }
}
```

Лимиты

- Объемы памяти:
  - адресное пр-во
  - стек
  - RSS
- Открытые файлы
- Количество процессов
- Время (процессорное!) работы процесса