Лекция 12
Процессы

Группы процессов

- Группа процессов выступает как единое целое при некоторых операциях (посылка сигнала, обработка Ctrl-C с терминала)
- Идентификатор группы процессов — pid первого процесса в группе
- Получение идентификатора группы:

```
pid_t getpgid(void);
```

Работа с группами процессов

int setpgid(pid_t pid, pid_t pgid);

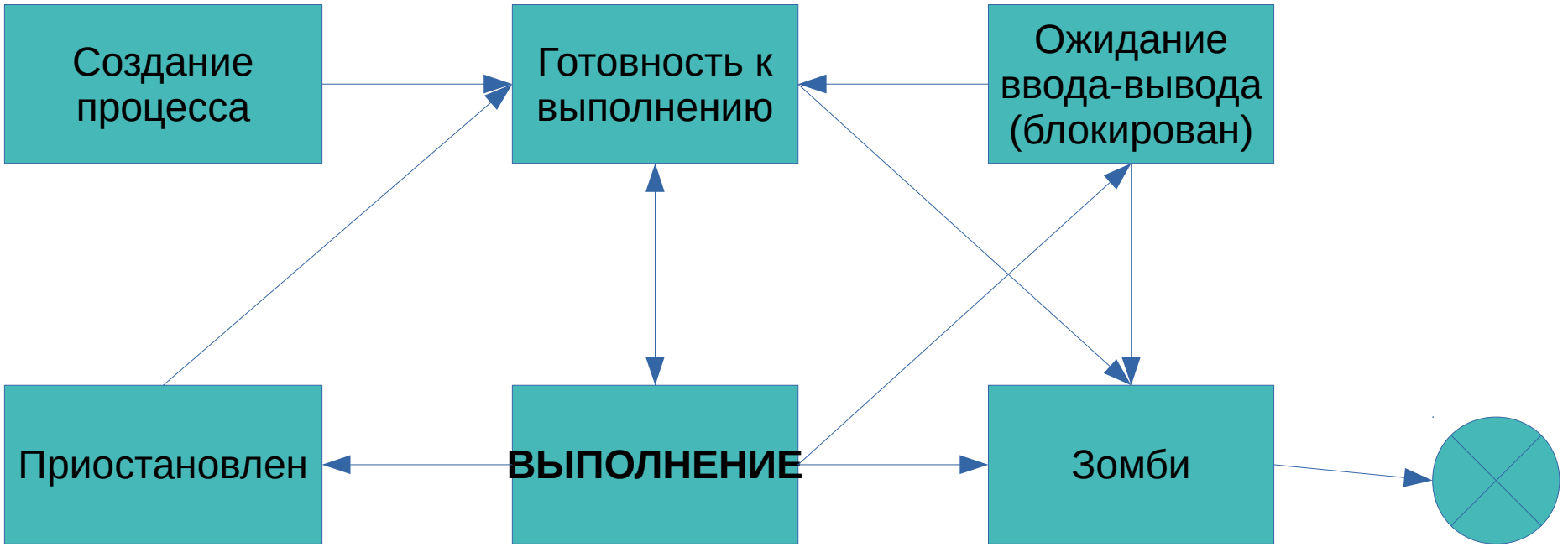
- setpgid(0, 0); - создается новая группа процессов, в которую помещается текущий процесс, pgid == getpid()
- setpgid(pid, 0); - создается новая группа процессов, в которую помещается указанный процесс, pgid == pid
- setpgid(0, pgid) — поместить текущий процесс в указанную группу процессов
- setpgid(pid, pgid) — поместить указанный процесс в указанную группу процессов

Системный вызов waitpid

int waitpid(int pid, int *pstatus, int flags);

- Допустимые значения pid сыновних процессов
  - < -1 — любой процесс из указанной группы
  - -1 — любой процесс
  - 0 — любой процесс из текущей группы
  - > 0 — указанный процесс
- Допустимые значения flags
  - WNOHANG — не блокировать процесс, если нет завершившихся сыновних процессов — в этом случае возвращается 0

Жизненный цикл процесса



Состояния процесса

- Готов к выполнению/выполняется (TASK_RUNNING) — ядро берет процессы из очереди готовых к выполнению и ставит на выполнение на процессор, обозначается 'R'
- Ожидает ввода-вывода (спит)
  - TASK_INTERRUPTIBLE ('S') - обычное ожидание, ожидание может быть прервано сигналом (в частности, завершения процесса)
  - TASK_UNINTERRUPTIBLE ('D') — непрерываемое ожидание, процесс не может быть убит. Обычно используется при операциях ввода-вывода с устройством, на котором размещена файловая система (диски)

Состояния процесса

- TASK_ZOMBIE ('Z') — ожидание опроса состояния завершения процесса
- TASK_STOPPED ('T') — процесс приостановлен, то есть не ждет i/o, но и не готов к выполнению. Либо приостановлен пользователем (Ctrl-Z с терминала), либо сигналом (SIGSTOP), либо попытался считать с терминала в фоновом режиме
- TASK_TRACED ('t') — процесс отлаживается

Состояния процессов

sleep(10);

Из состояния 'R' процесс переводится в состояние 'S', через 10 с процесс будет разбужен и переведен в состояние 'R'

read(0, buf, sizeof(buf)); // чтение из stdin

Если символов в буфере ввода нет, процесс переводится в состояние 'S' и будет оставаться в нем, пока не появятся данные, после чего будет переведен в состояние 'R'

read(fd, buf, sizeof(buf)); // чтение с диска

На время обмена с диском процесс находится в 'D'

Замещение тела процесса

- Замещение тела процесса — запуск на выполнение другого исполняемого файла в рамках текущего процесса
- Для замещения тела процесса используется семейство exec*: сист. вызов execve и функции execv, execvp, execl, execlp, execle
  - «v» - передается массив параметров
  - «l» - передается переменное число параметров
  - «e» - передается окружение
  - «p» - выполняется поиск по PATH

Системный вызов execve

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

- path — путь к исполняемому файлу
- argv — массив аргументов командной строки, заканчивается элементом NULL
- envp — массив переменных окружения, заканчивается элементом NULL
- Аргументы командной строки и переменные окружения помещаются на стек процесса
- При успехе системный вызов не возвращается

execve

- argv[0] обычно совпадает с path (но не обязано)
- Переменная 'environ' — текущее окружение процесса
- 'environ' должна быть объявлена явно, если требуется
- «Shebang» конструкции (#! /usr/bin/python) обрабатываются ядром

Сохранение атрибутов процесса

- Сохраняются все атрибуты, **за исключением**
  - Атрибутов, связанных с адресным пространством процесса
  - Сигналов, ожидающие доставки
  - Таймеров

Функция execlp

int execlp(const char *file, const char *arg, ...);

- Выполняется поиск исполняемого файла file по каталогам, перечисленным в переменной окружения PATH
- Аргументы запускаемого процесса передаются в качестве параметров функции execlp
- Последним аргументом функции должен быть NULL

Схема fork/exec

- Для запуска программ в отдельных процессах применяется комбинация fork/exec

- Системный вызов fork создает новый процесс
- В сыновнем процессе системными вызовами настраиваются параметры процесса (например, текущий рабочий каталог, перенаправления стандартных потоков и пр.)
- Вызовом exec* запускается требуемый исполняемый файл

Пример

```
int main(void)
{
  int pid, status, fd;
  pid = fork();
  if (!pid) {
    chdir("/usr/bin");
    fd = open("/tmp/log.txt", O_WRONLY|O_CREAT|O_TRUNC, 0600);
    dup2(fd, 1); close(fd);
    execlp("/bin/ls", "/bin/ls", "-l", NULL);
    fprintf(stderr, "Exec failed\n");
    _exit(1);
  }
  wait(&status);
  return 0;
}
```

Подготовка аргументов командной строки

- Часто необходимо запустить программу, если передана строка состоящая из имени программы и аргументов

int system(const char *command);

Например: res = system("ls -l");

Реализация с помощью execlp:
execlp("/bin/sh", "/bin/sh", "-c", command, NULL);

Привилегии процесса

- Уровень привилегий определяется его идентификаторами пользователя:
  - Uid (вызов getuid()) - реальный идентификатор пользователя, то есть идентификатор того, кто запустил программу
  - Effective UID (вызов geteuid()) - эффективный идентификатор, то есть с какими правами работает процесс
- Для групп — аналогично (getgid(), getegid())

Suid/sgid бит

- Исполняемые файлы могут изменять свои привилегии в специальных случаях
  - suid-бит (04000) — процесс запускается с euid **владельца исполняемого файла**, а не пользователя, запускающего процесс
  - sgid-бит (02000) — аналогично для группы
- seteuid(int uid) — установить новый эффективный идентификатор
  - Текущий euid сохраняется в saved uid и может быть потом восстановлен
  - Uid должен быть либо реальный идентификатор, либо сохраненный идентификатор
  - Можно переключаться между effective uid и real uid несколько раз
- setuid(int uid) — однократное переключение, устанавливает и реальный, и эффективный идентификатор
- Для групп аналогично (setegid, setgid)