Процессы. Часть 2

Лекция 11 по АКОС

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
```

Порождение процесса

```
int main() {
  pid_t  result = fork();
  if (-1==result) { perror("fork :-("); exit(1); }
  if (0==result) {
    // дочерний процесс
  }
  else {
    // родительский процесс
  }
}
```

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);
  }
}
```

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);
  }
}
```

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

Примеры SUID-программ

- sudo
- su
- sshd

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);
  }
}
```

Трассировка процессов *(Linux/BSD)*

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

    ptrace(PTRACE_TRACEME, 0, 0, 0);

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

Трассировка процессов *(Linux/BSD)*

```
. . .
else {
  // в родительском процессе
  int wstatus = 0;
  struct user_regs_struct state;
  bool stop = false;
  while (!stop) {
    ptrace(PTRACE_SYSCALL, pid, 0, 0);
    waitpid(pid, &wstatus, 0);
    stop = WIFEXITED(wstatus);
    if (WIFSTOPPED(wstatus)) {
      ptrace(PT_GETREGS, pid, 0, &state);
      fprintf(stderr, "syscall %lld\n", state.orig_rax);
    }
  }
}
```

Интроспекция поведения
ptrace

- Полноценная трассировка возможна не на всех процессорных архитектурах
- Трассировка используется отладчиком gdb и инструментом strace
- Позволяет творить и исследуемой программой всё что угодно (модифицировать память и регистры)

Интроспекция поведения *подмена функций линковщиком*

- Опция линковщика `--wrap=ИМЯ`
- Для gcc: `-Wl,--wrap=ИМЯ`
- Подмена функции на стадии линковки

```
pid_t __real_fork();
pid_t __wrap_fork()
{
    pid_t pid = __real_fork();
    if (pid > 0) {
        int scale = 1000 * 100; // quantum is 1/10 sec
        int quants = rand() % 10;
        usleep(scale * quants);
    }
    return pid;
}
```

Интроспекция поведения

подмена функций загрузчиком `ld-linux.so`

- Переменная окружения LD_PRELOAD
- Указанная библиотека функций имеет более высокий приоритет, чем glibc
- Может применяться в ситуациях, когда нет исходных текстов