СОЗДАНИЕ ПРОЦЕССОВ И ИСПОЛНЕНИЕ ПРОГРАММ

Системные вызовы и библиотеки Unix SVR4

Иртегов Д.В.

ФФ/ФИТ НГУ

Электронный лекционный курс подготовлен в рамках реализации

Программы развития НИУ-НГУ на 2009-2018 г.г.

ЦЕЛИ РАЗДЕЛА

По завершении этого раздела вы сможете:

- Создавать подпроцессы
- Исполнять новую программу
- Завершать процесс
- Ожидать завершения подпроцесса

ЧТО ТАКОЕ ПРОЦЕСС?

- процесс представляет собой исполняющуюся программу вместе с необходимым окружением (сегменты данных и стека, user area и др.)
- образ процесса – виртуальное адресное пространства процесса во время исполнения
- процесс не является программой

fork (2)

ИСПОЛЬЗОВАНИЕ

```
#include <sys/types.h>
#include <unistd.h>
pid_t fork(void);
```

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ

успех –

- родительский процесс: идентификатор порожденного процесса;
- порожденный процесс : 0

неуспех - -1 и errno установлена

Что делает fork

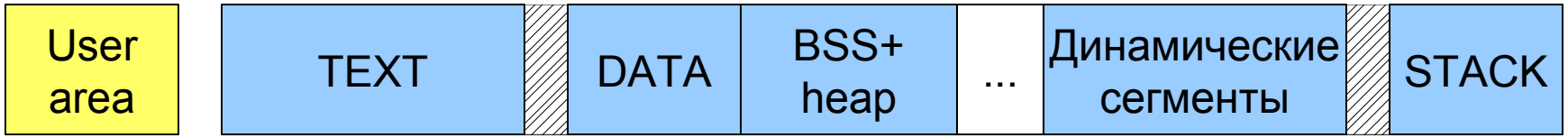
- Создает новый процесс
- Наследуются все «ручки» открытых файлов и ряд других параметров (ограничения, pgid, sid, uid, gid)
- Наследуется все адресное пространство родительского процесса
- При первой модификации страницы памяти происходит ее физическое копирование (Copy on Write, CoW)

Что делает fork

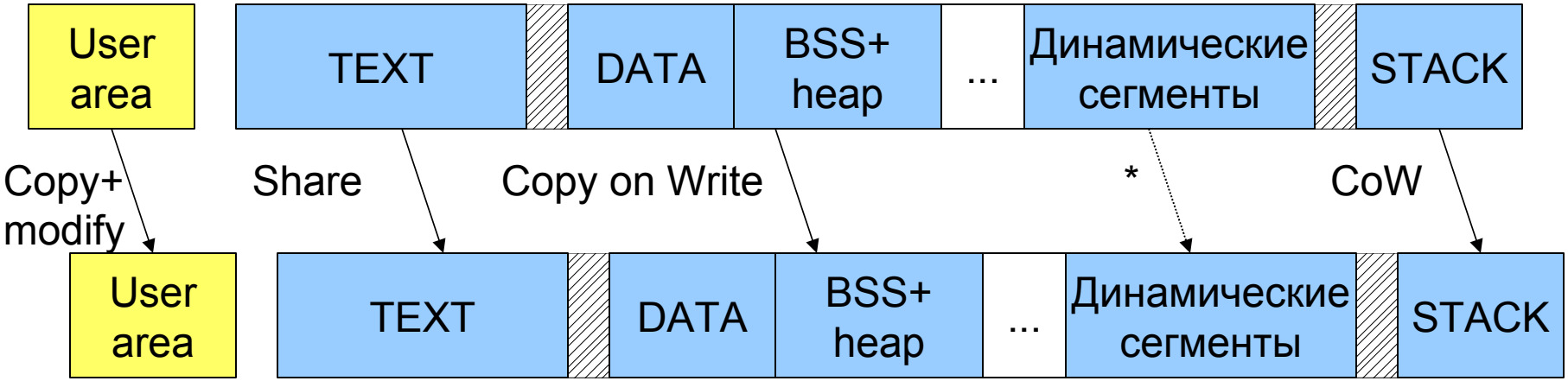
- НЕ наследуются:
  - pid
  - ppid
  - Захваченные участки файлов
  - Нити исполнения (кроме той, которая вызвала fork)

Что происходит при fork(2)

До:



После:



* - в зависимости от атрибутов, динамические сегменты могут быть разделяемые или CoW

Copy on Write

Система создает копию страницы при первой модификации

Немодифицированные страницы остаются разделяемыми

Обычно реализовано через защиту от записи на уровне MMU

При обращении к странице на чтение, нормальное обращение

При обращении на запись как родителем, так и потомком, исключение защиты памяти (GPF у x86)

ОС понимает, что это не настоящая ошибка защиты, и создает копию страницы

Сегменты, отображенные на память с флагом MAP_SHARED остаются разделямыми, CoW распространяется только на сегменты MAP_PRIVATE и MAP_ANON

Пример fork

```
1 #include <sys/types.h>
2 #include <unistd.h>
3 #include <stdio.h>
4
5 main()
6 {
7
8 printf("[%ld] parent process id: %ld\n",
9  getpid(), getppid());
10
11 fork();
12
13 printf("\n\t[%ld] parent process id: %ld\n",
14 getpid(), getppid());
15 }
```

Результат исполнения

```
bash> fork0
[18607] parent process id: 18606
[18608] parent process id: 18607
[18607] parent process id: 18606
```

exec (2)

ИСПОЛЬЗОВАНИЕ

```
#include <unistd.h>
int execl (const char *path, const char *arg0, ...,
  const char *argn, (char *) 0);
int execv (const char *path, char *const *argv);
int execle (const char *path, const char *arg0, ...
  /* const char *argn, (char *) 0,
  const char *envp[] */);
int execve (const char *path, char *const *argv,
  char *const *cnup);
int execlp (const char *file, const char *arg0, ...
  /* const char *argn, (char *) 0 */);
int execvp (const char *file, char *const *argv);
```

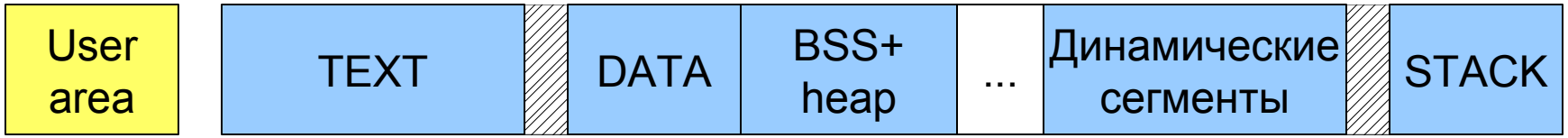
ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ

успех - не возвращает управление

неуспех - -1 и errno установлена

Что делает exec?

До:



После:

Что делает exec

- Наследуется:
  - Почти все содержимое user area: pid, ppid, ruid, rgid, ограничения rlimit, и др.
    - Поэтому говорят, что при exec происходит замена программы в том же процессе
  - Открытые файлы (кроме CLOEXEC)
  - Захваченные участки файлов
  - euid/egid (если исполняемый файл не setuid/setgid)

Что делает exec

- НЕ наследуются
  - Адресное пространство и отображенные на память файлы
  - Пользовательские обработчики сигналов
  - Регистры ЦПУ (контекст процесса)
  - Нити исполнения
  - euid/egid, если исполняемый файл setuid/setgid
- Копируются
  - Аргументы exec: argv и environ.

exit (2)

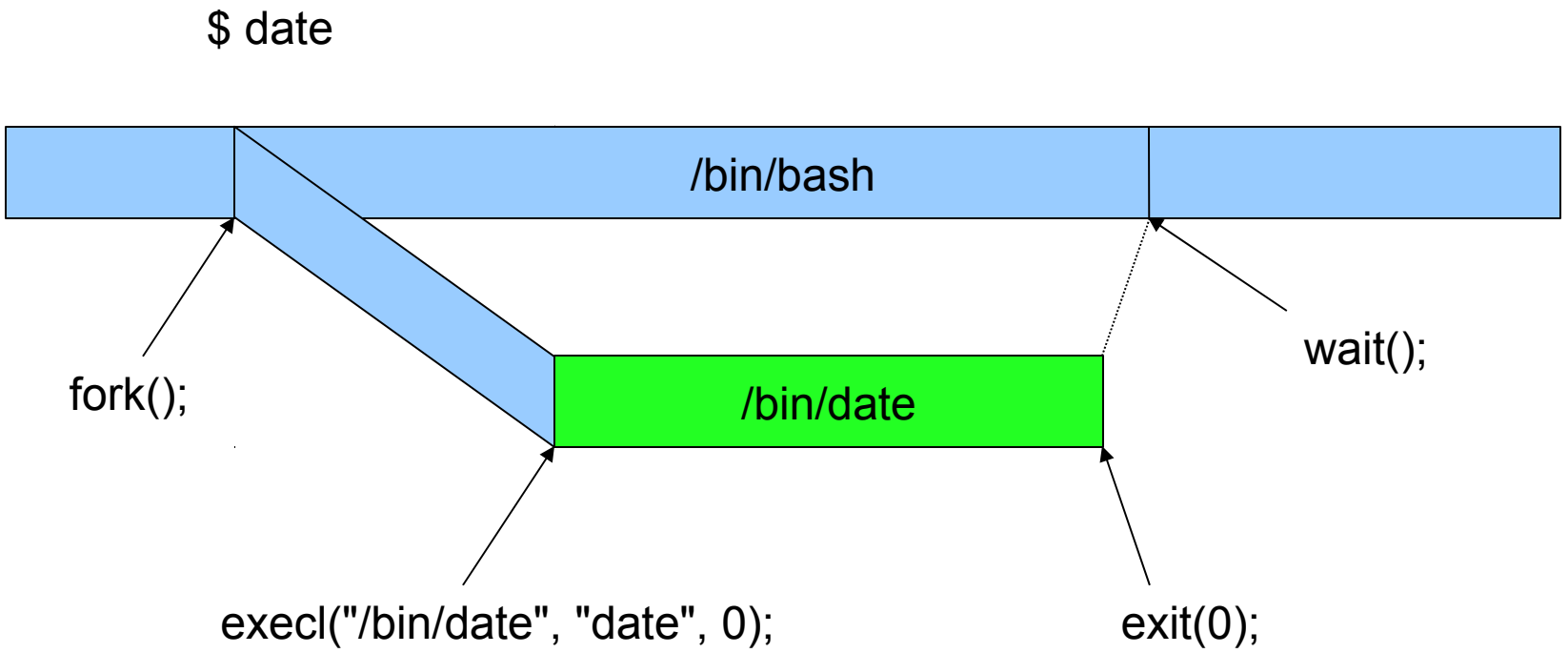
ИСПОЛЬЗОВАНИЕ

```
#include <stdlib.h>
void exit (int status);
```

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ

нет - никогда не возвращает управление

Исполнение программ shell

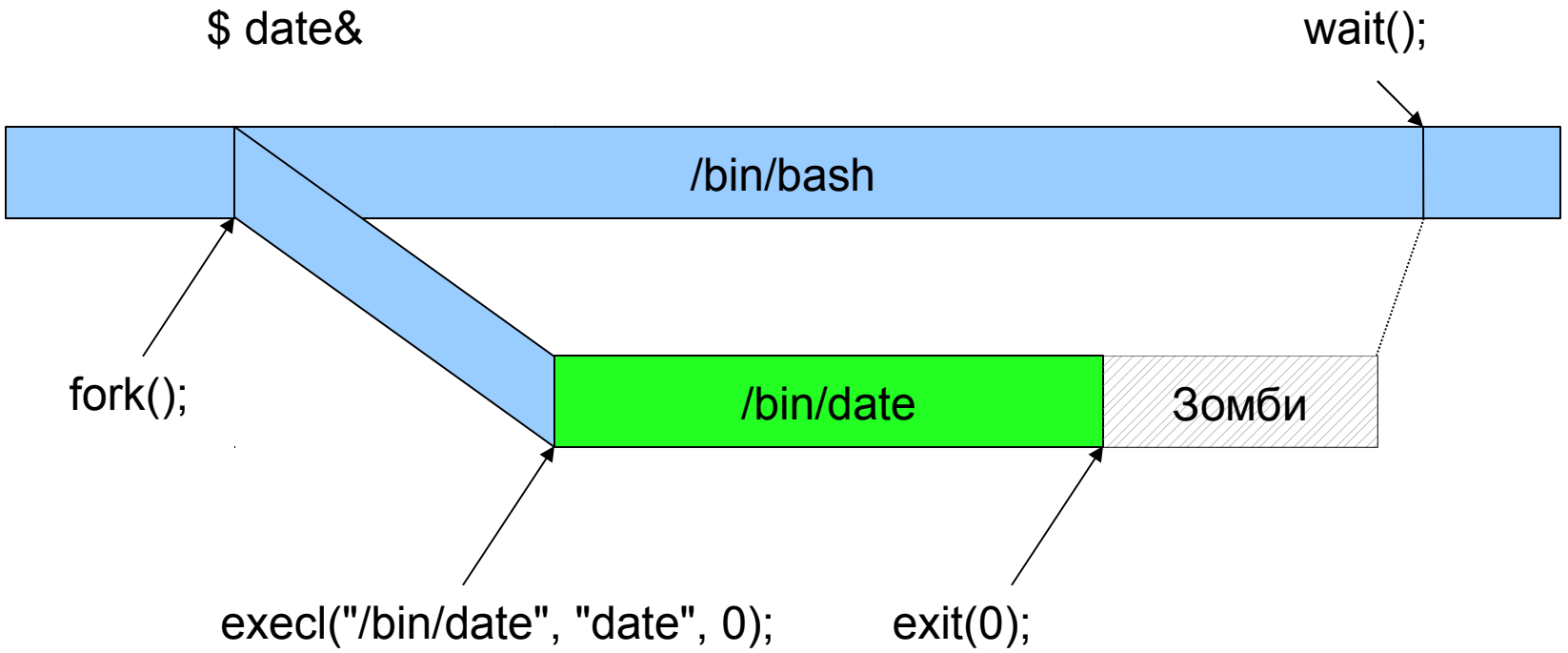


Аргументы-шаблоны fnmatch(5) заменяются на списки аргументов:

*.c → список всех файлов в текущей директории с именами, оканчивающимися на .c

*/[Mm]akefile → список всех файлов Makefile или makefile во всех поддиректориях текущей директории

Процессы-зомби



- Занимает pid
- Хранит статус завершения процесса, пока родитель его не прочитает

Сигналы

- Средство обработки ошибок и исключений
  - Деление на ноль (SIGFPE)
  - Ошибка защиты памяти (SIGSEGV)
  - Прерывание пользователем (SIGINT/SIGQUIT)
  - Завершение терминальной сессии (SIGHUP)
- Отправляются ядром
- Могут отправляться программно: kill(2), kill(1)
- Необработанные сигналы приводят к завершению процесса
- Подробно рассматриваются в разделе «Сигналы»

wait(2)

ИСПОЛЬЗОВАНИЕ

```
#include <sys/types.h>
#include <wait.h>
pid_t wait(int *stat_loc);
```

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ

успех - идентификатор завершившегося подпроцесса

неуспех - -1 и errno установлена

Формат статуса wait

	Параметр exit	Номер сигнала

wstat(2)

ИСПОЛЬЗОВАНИЕ

```
#include <sys/wait.h>
```

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ

`WIFEXITED(stat)` ненулевое значение, если подпроцесс нормально завершился

`WEXITSTATUS(stat)` код завершения подпроцесса

`WIFSIGNALED(stat)` ненулевое значение, если подпроцесс был принудительно завершен сигналом

`WTERMSIG(stat)` номер сигнала, который вызвал завершение подпроцесса

`WCOREDUMP(stat)` если WIFSIGNALED - не ноль, то возвращает ненулевое значение, если был создан образ ядра

waitid(2)

ИСПОЛЬЗОВАНИЕ

```
#include <sys/types.h>
#include <wait.h>
pid_t waitid (idtype_t idtype, id_t id,
  siginfo_t *infop, int options);
```

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ

успех – pid процесса

неуспех - -1 и errno установлена

Флаги waitid

`WEXITED` ожидать нормального завершения подпроцесса (по exit(2)).

`WTRAPPED` ожидать прерывания трассировки или точки останова в отлаживаемом процессе (ptrace(2)).

`WSTOPPED` ожидать, пока подпроцесс будет остановлен получением сигнала.

`WCONTINUED` ожидать, пока остановленный подпроцесс не начнет исполнение.

`WNOHANG` немедленно возвращать управление, если нет немедленно доступного слова состояния

`WNOWAIT` неразрушающий wait (зомби-запись процесса не уничтожается)

waitpid(2)

ИМЯ

waitpid - ожидание изменения состояния процесса

ИСПОЛЬЗОВАНИЕ

```
#include <sys/types.h>
#include <sys/wait.h>
pid_t waitpid(pid_t pid, int *stat_loc,
                int options);
```

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ

успех - идентификатор подпроцесса или 0, если WHOHANG

неуспех - -1 и errno установлена

Значения pid

-1 — для всех подпроцессов

>0 — для подпроцесса с идентификатором pid

0 — для любого подпроцесса с тем же pgid, что у вызывающего процесса

<-1 — для любого подпроцесса, чей идентификатор группы процессов равен -pid

Значения flags

`WNOHANG` то же, что и для waitid(2).

`WUNTRACED` то же, что WSTOPPED для waitid(2).

`WCONTINUED` то же, что и для waitid(2).

`WNOWAIT` то же, что и для waitid(2).

atexit (2)

ИСПОЛЬЗОВАНИЕ

```
#include <stdlib.h>
int atexit (void (*func)(void));
```

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ

успех - 0

неуспех - ненулевое значение