Technology Tarena 达内科技

Unix系统高级编程

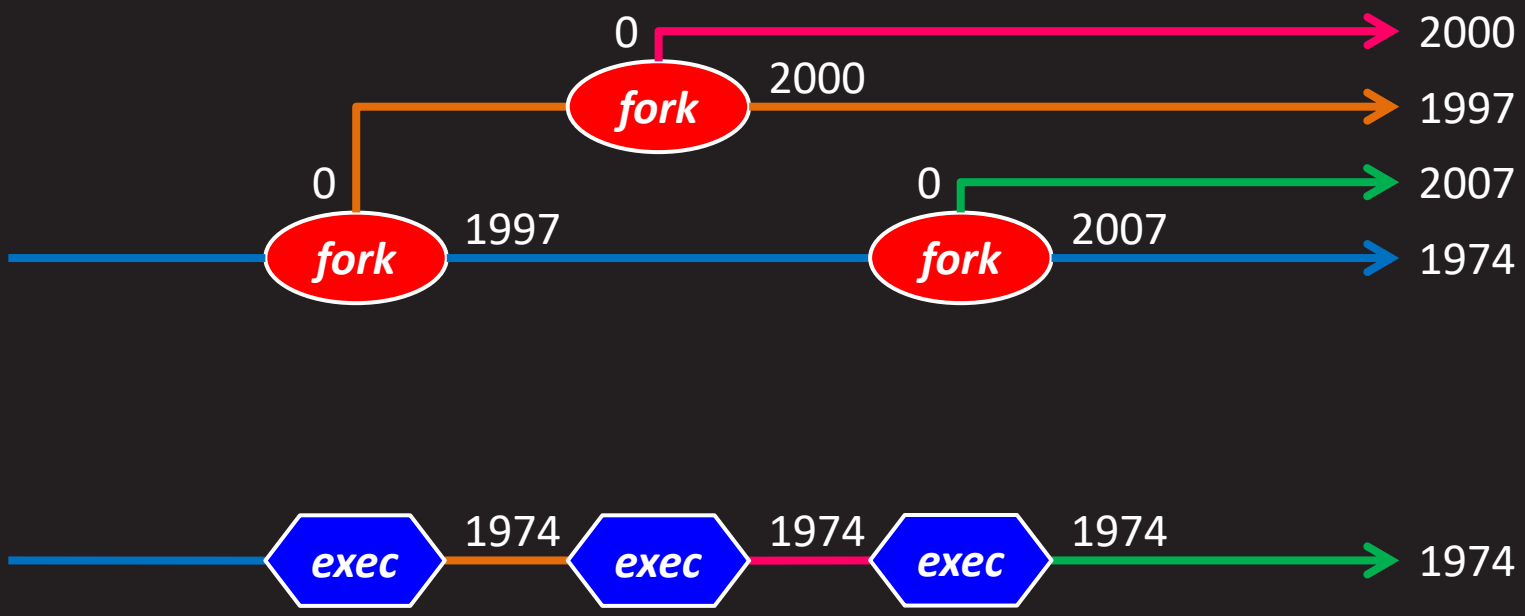
创建新进程 Unit16

创建新进程

exec

exec

- 与fork或vfork函数不同，exec函数不是创建调用进程的子进程，而是创建一个新的进程取代调用进程自身。新进程会用自己的全部地址空间，覆盖调用进程的地址空间，但进程的PID保持不变



exec（续1）

- exec不是一个函数而是一堆函数(共6个)，一般称为exec函数族。它们的功能是一样的，用法也很相近，只是参数的形式和数量略有不同

知识讲解

```
#include <unistd.h>

int execl (const char* path, const char* arg, ...);
int execlp (const char* file, const char* arg, ...);
int execle (const char* path, const char* arg, ...,
    char* const envp[]);
int execv (const char* path, char* const argv[]);
int execvp (const char* file, char* const argv[]);
int execve (const char* path, char* const argv[],
    char* const envp[]);
```

成功不返回，失败返回-1

exec（续2）

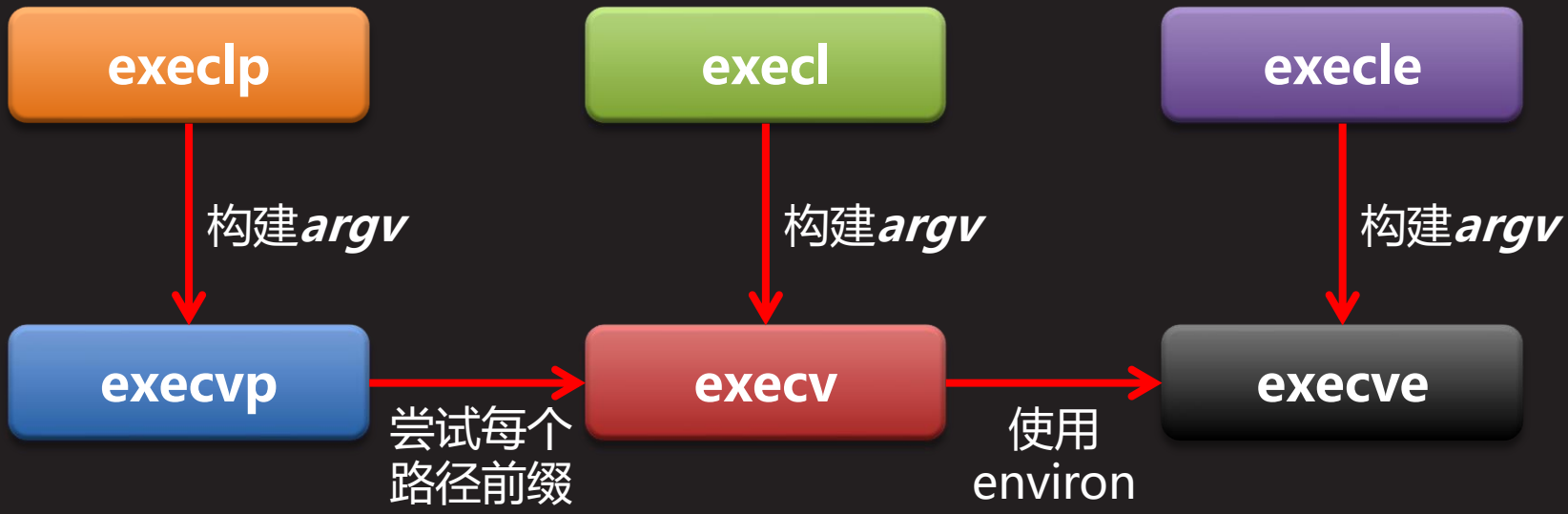
- exec函数族一共包括6个函数，它们的函数名都是在exec后面加上一到两个字符后缀，不同的字符后缀代表不同的含义
  - `l`：即list，新进程的命令行参数以字符指针列表(const char* arg, ...)的形式传入，列表以空指针结束
  - `p`：即path，若第一个参数中不包含“/”，则将其视为文件名，并根据PATH环境变量搜索该文件
  - `e`：即environment，新进程的环境变量以字符指针数组(char* const envp[])的形式传入，数组以空指针结束，不指定环境变量则从调用进程复制
  - `v`：即vector，新进程的命令行参数以字符指针数组(char* const argv[])的形式传入，数组以空指针结束

知识讲解

exec（续3）

- 其实6个exec函数中只有execve是真正的系统调用，其它5个函数不过是对execve函数的简单包装

知识讲解



exec（续4）

- 例如
  - 启动/bin/vi替换当前进程

    ```
    execl ("/bin/vi", "vi", NULL);
    ```

  - 用PATH环境变量作为搜索路径

    ```
    execlp ("vi", "vi", NULL);
    ```

  - 设置特殊的环境变量

    ```
    char* envp[] = {"HOME=/home/tarena", NULL};
    execle ("/bin/vi", "vi", NULL, envp);
    ```

  - 它们的v版本

    ```
    char* argv[] = {"vi", NULL};
    execv ("/bin/vi", argv);
    execvp ("vi", argv);
    execve ("/bin/vi", argv, envp);
    ```

知识讲解

exec（续5）

知识讲解

- 当在Shell下启动进程时，Shell会把路径的最后一个成分(通常是可执行程序的硬链接名)作为传给main函数的第一个命令行参数，即argv[0]。很多系统工具有不同的名称，其实它们不过是指向同一个程序的不同的硬链接。这些工具需要根据Shell传给main函数的第一个参数来判断用户使用的是哪个硬链接，以提供相应的功能。因此，调用exec函数时最好遵循Unix的习俗，用可执行程序的硬链接名作为新进程的第一个命令行参数，如上例中
  - execl ("/bin/vi", "vi", NULL);
  - execlp ("vi", "vi", NULL);
  - execle ("/bin/vi", "vi", NULL, envp);
  - char* argv[] = {"vi", NULL};

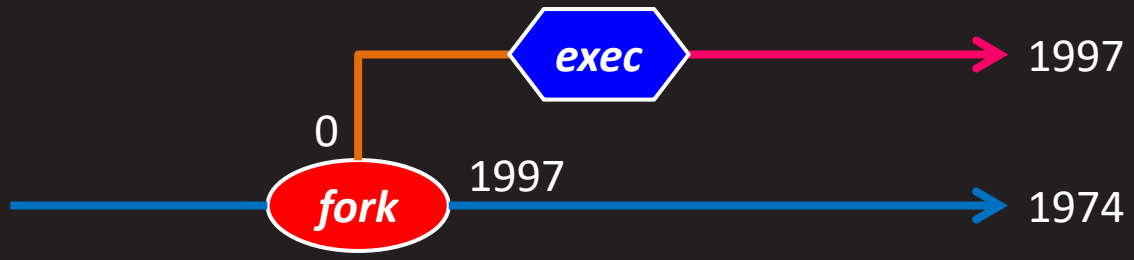
exec（续6）

- 调用exec函数不仅改变调用进程的地址空间和进程映像，调用进程的一些属性也发生了变化
  - 任何处于阻塞状态的信号都会丢失
  - 被设置为捕获的信号会还原为默认操作
  - 有关线程属性的设置会还原为缺省值
  - 有关进程的统计信息会复位
  - 与进程内存相关的任何数据都会丢失，包括内存映射文件
  - 标准库在用户空间维护的一切数据结构(如通过atexit或on_exit函数注册的退出处理函数)都会丢失
- 但也有些属性会被新进程继承下来，比如PID、PPID、实际用户ID和实际组ID、优先级，以及文件描述符等

知识讲解

exec（续7）

- 注意如果进程创建成功，exec函数是不会返回的，因为成功的exec调用会以跳转到新进程的入口地址作为结束，而刚刚运行的代码是不会存在于新进程的地址空间中的。但如果进程创建失败，exec函数会返回-1
- 调用exec函数固然可以创建出新的进程，但是新进程会取代原来的进程。如果既想创建新的进程，同时又希望原来的进程继续存在，则可以考虑fork+exec模式，即在fork产生的子进程里调用exec函数，新进程取代了子进程，但父进程依然存在



知识讲解

exec（续8）

- 例如
  - pid_t pid = fork ();
    if (pid == -1) {
    perror ("fork"); exit (EXIT_FAILURE); }
    if (pid == 0)
    if (execl ("ls", "ls", "-l", NULL) == -1) {
    perror ("execl"); exit (EXIT_FAILURE); }
- 事实上，在这种场合使用vfork更合适一些
  - pid_t pid = vfork ();
    if (pid == -1) {
    perror ("vfork"); exit (EXIT_FAILURE); }
    if (pid == 0)
    if (execl ("ls", "ls", "-l", NULL) == -1) {
    perror ("execl"); _exit (EXIT_FAILURE); }

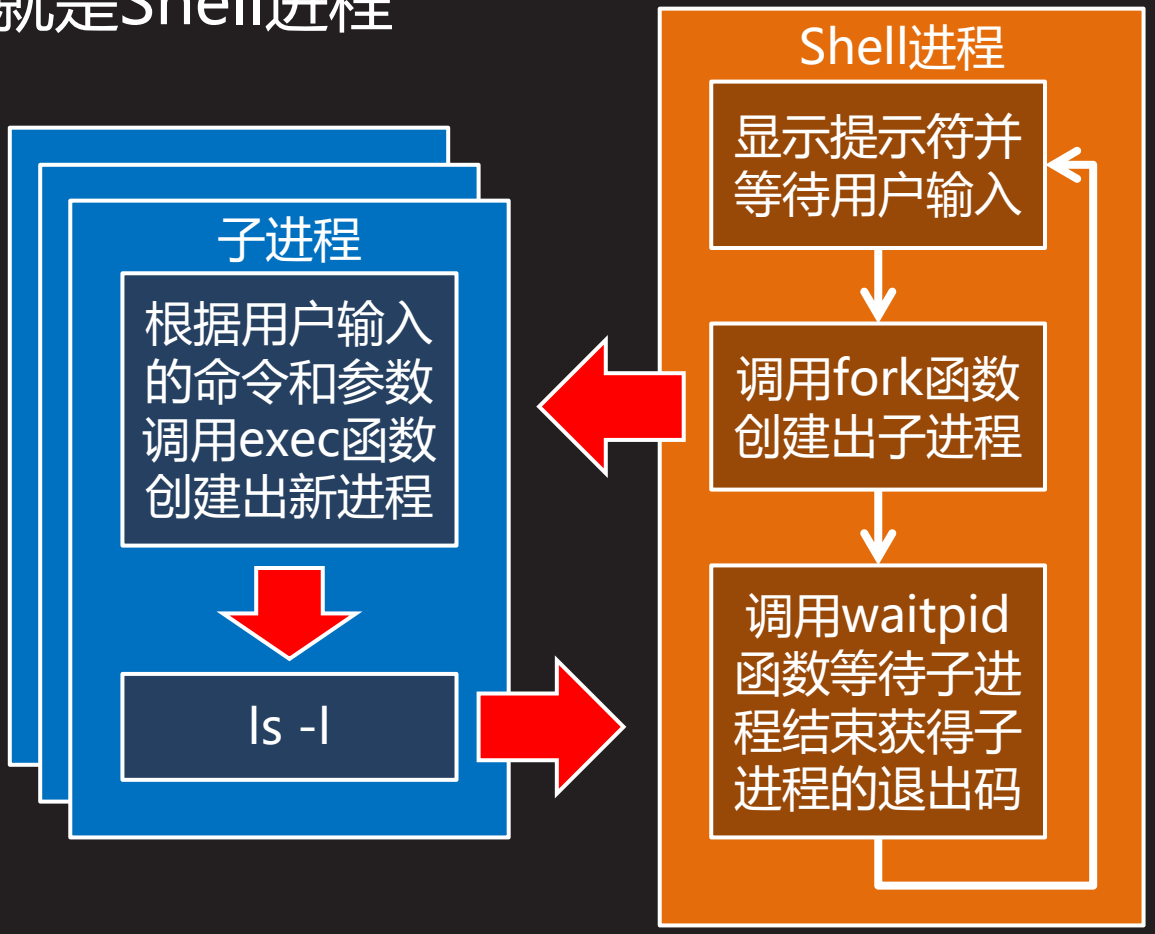
exec（续9）

- 如果原来的进程希望等待新进程结束以后再继续
  - pid_t pid = vfork ();
    if (pid == -1) {
    perror ("vfork"); exit (EXIT_FAILURE); }
    if (pid == 0)
    if (execl ("ls", "ls", "-l", NULL) == -1) {
    perror ("execl"); _exit (EXIT_FAILURE); }
    int status;
    if (waitpid (pid, &status, 0) == -1) {
    perror ("waitpid"); exit (EXIT_FAILURE); }
    if (WIFEXITED (status))
    printf ("%d进程正常终止，退出码%d\n",
    pid, WEXITSTATUS (status));
    else
    printf ("%d进程异常终止，终止信号%d\n",
    pid, WTERMSIG (status));

知识讲解

exec（续10）

- 如果一个进程可以根据用户的输入创建出不同的进程，并在所建进程结束以后继续重复这个过程，那么这个进程其实就是Shell进程



知识讲解

创建新进程

【参见：argenv.c、exec.c】

课堂练习

- 创建新进程

Technology Tarena 达内科技

system

system

- 执行Shell命令

```
#include <stdlib.h>

int system (const char* command);

                成功返回command进程的终止状态，失败返回-1
```

  - *command*：Shell命令行字符串
- system函数执行*command*参数所表示的命令行，并返回命令进程的终止状态
- 若*command*参数取NULL，返回非0表示Shell可用，返回0表示Shell不可用

知识讲解

system（续1）

- 在system函数内部调用了vfork、exec和waitpid等函数
  - 如果调用vfork或waitpid函数出错，则返回-1
  - 如果调用exec函数出错，则在子进程中执行exit(127)
  - 如果都成功，则返回***command***进程的终止状态(由waitpid的***status***参数获得)
- 使用system函数而不用vfork+exec的好处是，system函数针对各种错误和信号都做了必要的处理，而且system是标准库函数，可跨平台使用

知识讲解

system（续2）

- 例如
  - int status;
    if ((status = system (NULL)) == -1) {
        perror ("system"); exit (EXIT_FAILURE); }
    if (! status)
        printf ("Shell不可用\n");
    else {
        if ((status = system ("ls -l")) == -1) {
            perror ("system"); exit (EXIT_FAILURE); }
        printf ("%d\n", WEXITSTATUS (status));
    }

知识讲解

执行命令

【参见：system.c】

课堂练习

- 执行命令

Technology Tarena 达内科技

总结和答疑