多进程编程

进程的定义和特征

- 进程的定义
- 进程是一个具有一定独立功能的程序关于某个数据集合的一次运行活动。
- 进程的实体结构
- （１）进程控制块（ PCB ）
- （２）程序段
- （３）数据段

进程控制块

- 进程控制块是进程实体的一部分，是操作系统中最重要的记录型数据结构。 PCB 中记录了操作系统所需的，用于描述进程进展情况及控制进程运行所需的全部信息。
- PCB 是进程存在的惟一标志。
- 一般把 PCB 存放在操作系统专门开辟的 PCB 区内。
- 在进程控制块中，主要包括下述 4 方面的信息。
- （ 1 ）进程描述信息
- 进程标识符。 每个进程都有惟一的进程标识符，用以识别不同的进程。
- 用户名或用户标识号。 每个进程都隶属于某个用户，有利于资源共享与保护。
- 家族关系。 标识进程之间的家族关系。

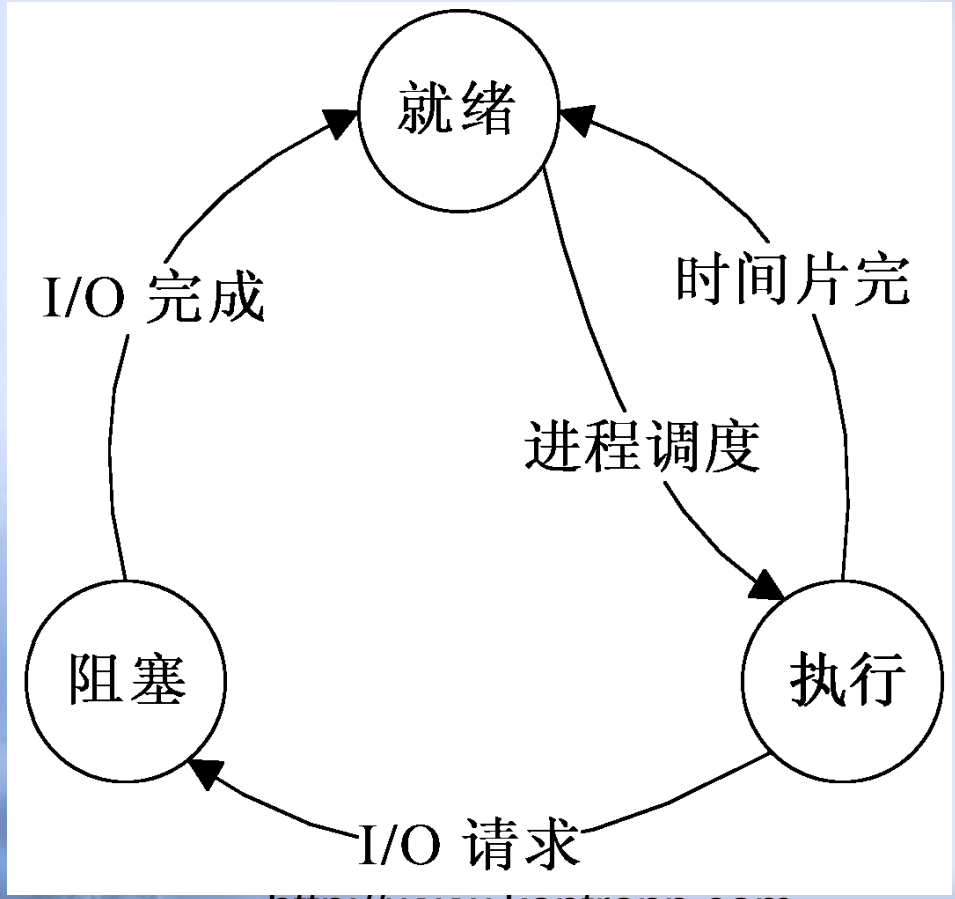
进程控制块

- （２）处理机状态信息
- 通用寄存器、指令计数器、程序状态字（ PSW ）、用户栈指针等
- （３）进程调度信息
- 进程状态。指明进程的当前状态，以作为进程调度和进程对换时的依据。
- 进程优先级。用于描述进程使用处理机的优先级别的一个整数，优先级别高的进程先获得处理机。
- 进程调度所需的其他信息。如进程已等待 CPU 的时间总和、进程已执行的时间总和等。
- 事件。指进程被阻塞的原因。
- （４）进程控制信息
- 程序和数据的地址。指出该进程的程序和数据所在的内存或外存地址，以便再调度到该进程执行时，能从中找到其程序和数据。
- 进程同步和通信机制。指实现进程同步和进程通信时所必须的机制，如消息队列指针、信号量等。这些数据应全部或部分地存放在 PCB 中。

进程的状态及其转换

- 1．进程的基本状态
- （1）就绪状态
- 当进程已分配到除处理机以外的所有必要的资源后，只要再获得处理机便可立即执行，这时进程的状态称为就绪状态。
- （2）执行状态
- 执行状态是指进程已获得处理机、其程序正在执行的状态。
- （3）阻塞状态
- 正在执行的进程因发生某事件而暂时无法继续执行时，便放弃处理机而处于暂停状态，这种暂停状态被称为阻塞状态。

进程的状态及其转换



获取进程标识

- #include <sys/types.h>
- #include <unistd.h>
- pid_t getpid(void);
- 功能：获取当前进程 ID
- 返回：调用进程的进程 ID

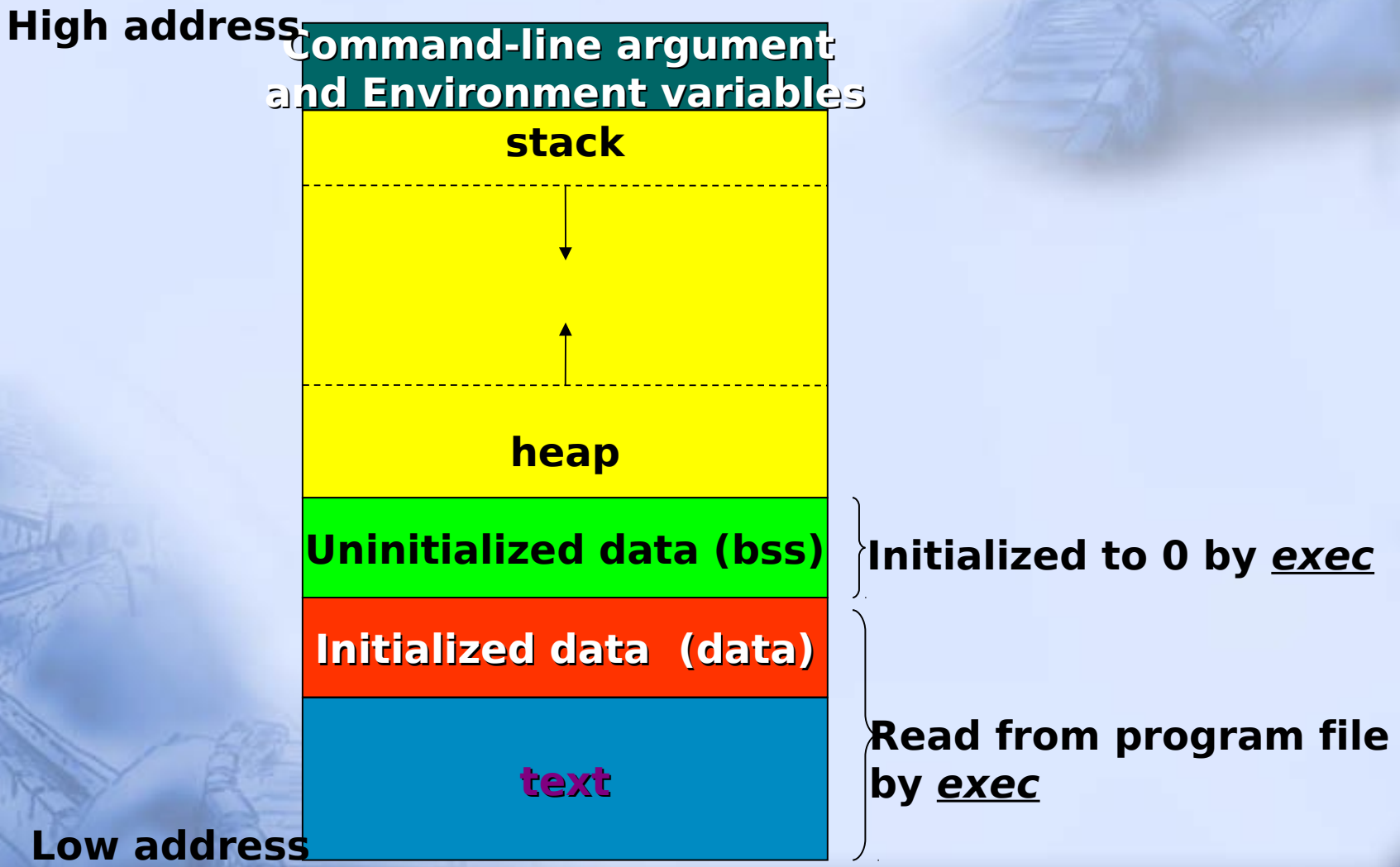
- #include <sys/types.h>
- #include <unistd.h>
- pid_t getppid(void);
- 功能：获取父进程 ID
- 返回：调用进程的父进程 ID

命令行参数

两种获得命令行参数的写法：

- main(int argc, char *argv[]);
- main(int argc, char *argv[], char *envp[]);

C 进程的内存分布图


High address
Command-line argument and Environment variables
stack
heap
Uninitialized data (bss)
Initialized to 0 by exec
Initialized data (data)
text
Read from program file by exec
Low address

进程创建

- #include <sys/types.h>
- #include <unistd.h>
- pid_t fork(void);
- 功能：创建一个新的进程。
- 返回：子进程中为 0，父进程中为子进程 ID，出错为 -1
- 说明：
  - 由 fork 创建的新进程被称为子进程（ child process ）。
  - 父、子进程完全一样（代码、数据），子进程从 fork 内部开始执行；父进程 fork 返回子进程的 pid 后，接着执行下一条语句
  - 该函数被调用一次，但返回两次。两次返回的区别是子进程的返回值是 0，而父进程的返回值则是子进程的进程 ID。
  - 一般来说，在 fork 之后是父进程先执行还是子进程先执行是不确定的。这取决于内核所使用的调度算法。

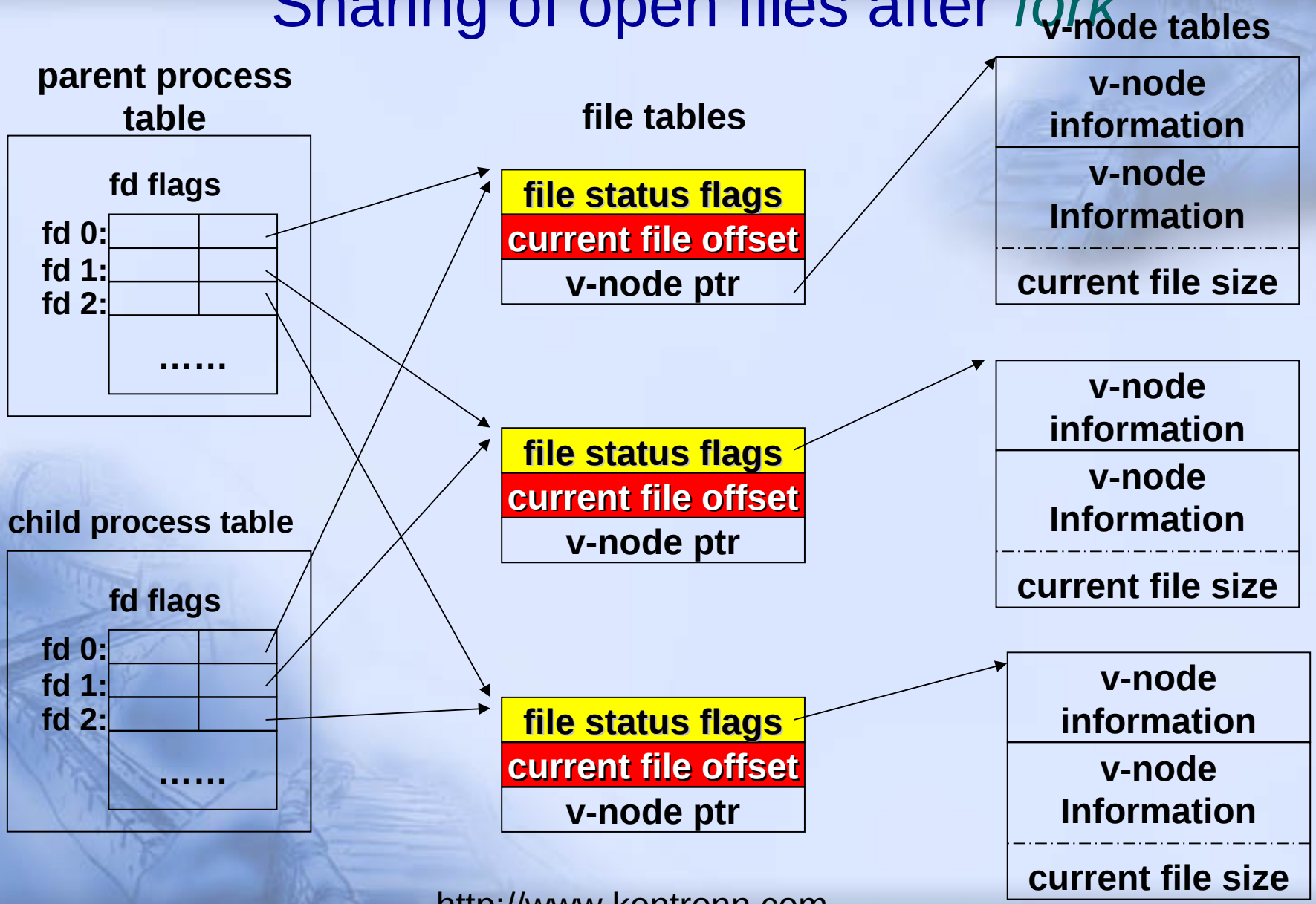
父、子进程之间的继承

- 使用 fork 函数得到的子进程从父进程处继承了整个进程的地址空间，包括：进程上下文、进程堆栈、内存信息、打开的文件描述符、信号控制设置、进程优先级、进程组号、当前工作目录、根目录、资源限制、控制终端等。

父、子进程之间的区别

- fork 的返回值；
- 进程 ID 、不同的父进程 ID ；
- 父进程设置的锁，子进程不继承；
- 子进程的未决告警被清除；
- 子进程的未决信号集设置为空集。

Sharing of open files after *fork*



wait 和 waitpid 函数

- 当一个进程正常或异常终止时，内核就向其父进程发送 SIGCHLD 信号。因为子进程终止是个异步事件 (这可以在父进程运行的任何时候发生)，所以这种信号也是内核向父进程发的异步通知。父进程可以忽略该信号，或者提供一个该信号发生时即被调用执行的函数 (信号处理程序)。对于这种信号的系统默认动作是忽略它。
- wait 函数用于使父进程阻塞，直到一个子进程结束或者该进程接收到一个指定信号为止。

wait 和 waitpid 函数

- 父进程调用 wait 或 waitpid ， 该父进程可能会：
  - 阻塞 (如果其所有子进程都还在运行)。
  - 带子进程的终止状态立即返回 (如果一个子进程已终止，正等待父进程存取其终止状态)。
  - 出错立即返回 (如果它没有任何子进程)。

wait 和 waitpid 函数

- #include <sys/types.h>
- #include <sys/wait.h>
- pid_t wait(int * status) ;
- pid_t waitpid(pid_t pid, int * status, int options) ;
- 功能：等待进程。
- 返回值：若成功则为子进程 ID 号，若出错则为 -1.
- 参数说明：
- status ： 用于存放进程结束状态。
- pid ： 要等待的进程 ID 。
  - pid == -1 等待任一子进程。于是在这一功能方面 waitpid 与 wait 等效。
  - pid > 0 等待其进程 ID 与 PID 相等的子进程。
  - pid == 0 等待其组 ID 等于调用进程的组 ID 的任一子进程。
  - pid < -1 等待其组 ID 等于 PID 的绝对值的任一子进程。
- options ： 设置等待方式。
  - 0 ： 不设置。
  - WNOHANG ： 如果没有任何已经结束的进程则马上返回，不等待。
  - WUNTRACED ： 如果子进程进入暂停状态则马上返回。

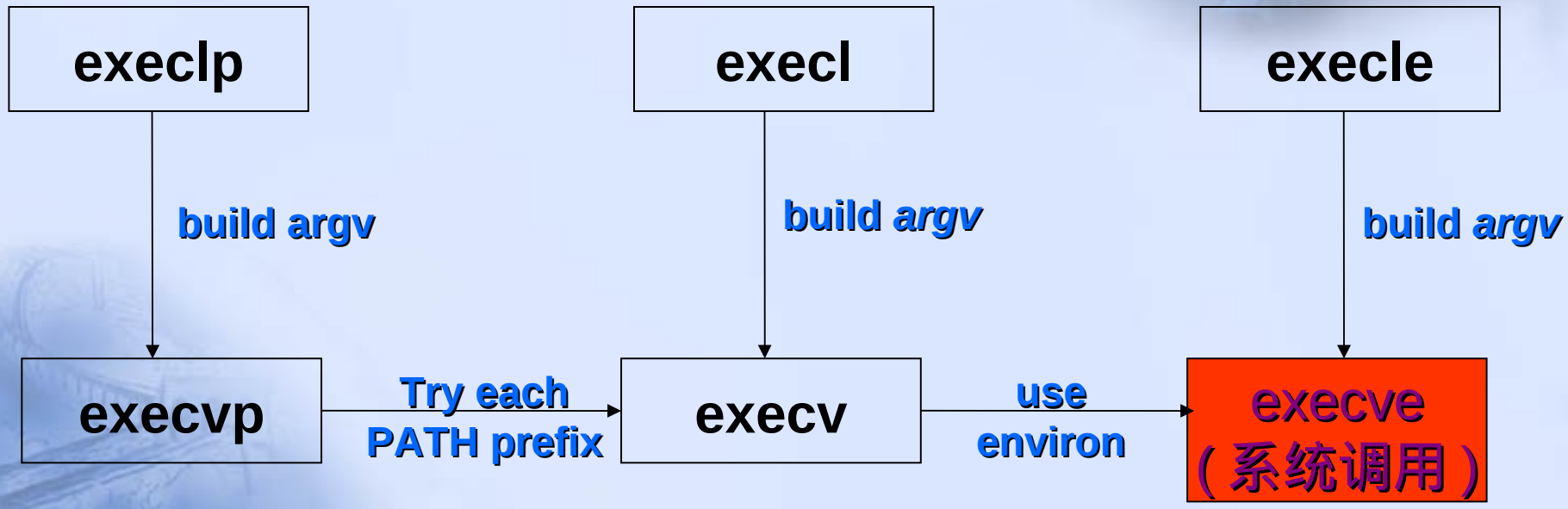
exec 函数

- 在用 fork 函数创建子进程后，子进程往往要调用一个 exec 函数以执行另一个程序。
- 当进程调用一种 exec 函数时，该进程完全由新程序代换，而新程序则从其 main 函数开始执行。因为调用 exec 并不创建新进程，所以前后的进程 ID 并未改变。 exec 只是用另一个新程序替换了当前进程的正文、数据、堆和栈段。

exec 函数

- #include <unistd.h>
- int execl(const char * pathname, const char * arg0, ... /* (char *) 0 */);
- int execv(const char * pathname, char *const argv []);
- int execle(const char * pathname, const char * arg0, .../* (char *)0, char *const envp [] */);
- int execve(const char * pathname char *const argv [], char *const envp []);
- int execlp(const char * filename, const char * arg0, ... /* (char *) 0 */);
- int execvp(const char * filename, char *const argv []);
- 功能：实现代码替换
- 返回值：若出错则为 - 1，若成功替换新代码。

Relationship of the six *exec*



exec 函数

- E: 指可以传递环境变量表
- L: 单独的参数传递，最后要有一个 NULL
- V: 传一个指针数组名
- P: 按照环境变量来查找

程序举例：

- char *ps_argv[]={“ps”,”-ax”, NULL};
- char *ps_envp[]={“PATH=/bin:/usr/bin”,”TERM=console”, NULL}
- execl(“/bin/ps”, “ps”, “-ax”, NULL);
- execv(“/bin/ps”, ps_argv);
- execle(“/bin/ps”, “ps”, “-ax”, NULL, ps_envp);
- execve(“/bin/ps”, ps_argv, ps_envp);
- execlp(“ps”, “ps”, “-ax”, NULL);
- execvp(“ps”, ps_argv);