Technology Tarena 达内科技

Unix系统高级编程

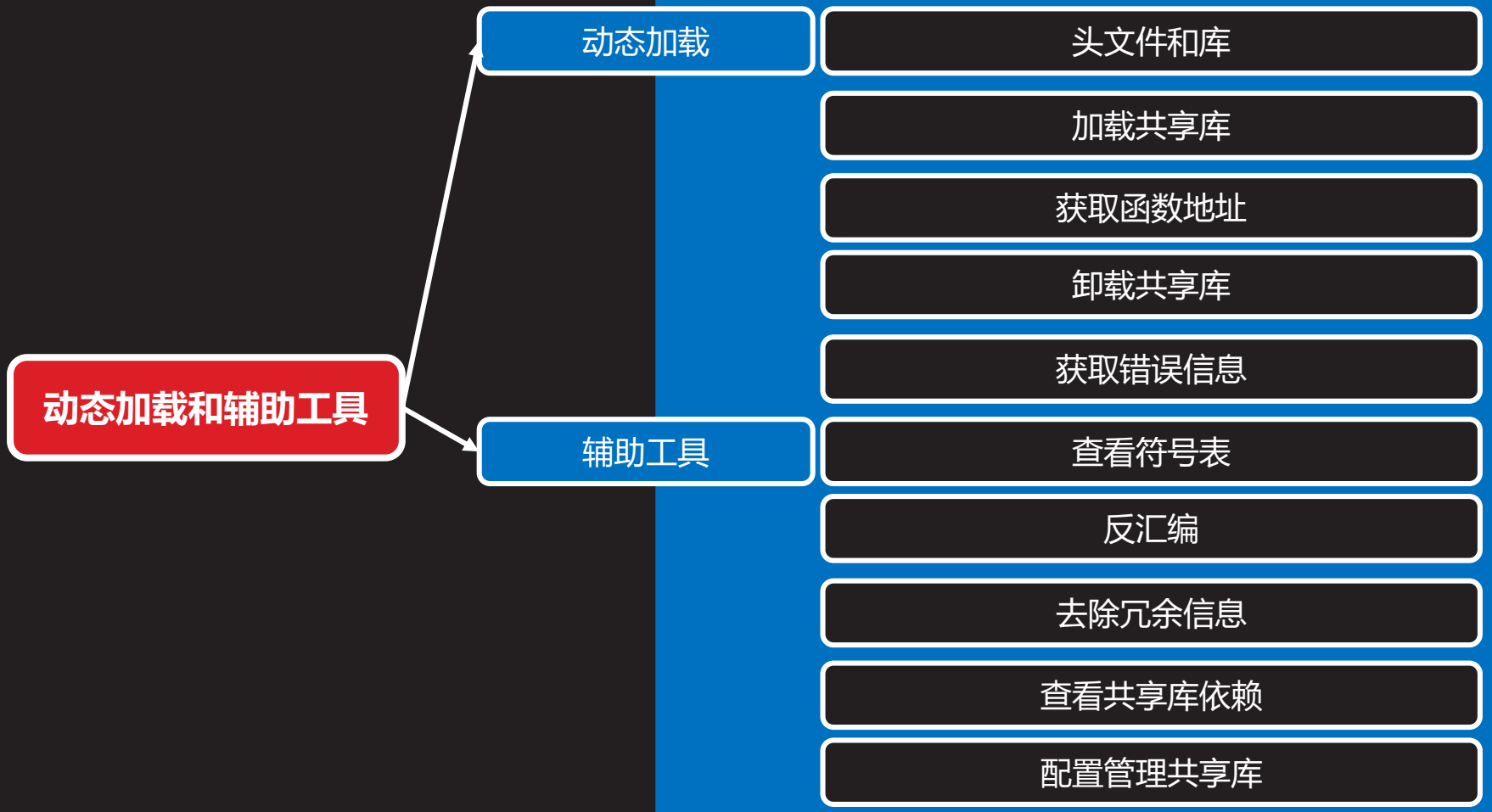
PART 1

DAY02

内容

上午	09:00 ~ 09:30	作业讲解和回顾
	09:30 ~ 10:20	动态加载和辅助工具
	10:30 ~ 11:20	
	11:30 ~ 12:20	错误处理
下午	14:00 ~ 14:50	
	15:00 ~ 15:50	环境变量
	16:00 ~ 16:50	
	17:00 ~ 17:30	总结和答疑

动态加载和辅助工具


动态加载和辅助工具
动态加载
辅助工具
头文件和库
加载共享库
获取函数地址
卸载共享库
获取错误信息
查看符号表
反汇编
去除冗余信息
查看共享库依赖
配置管理共享库

Technology Tarena 达内科技

动态加载

头文件和库

- 在程序中动态加载共享库需要调用一组特殊的函数，它们被声明于一个专门的头文件，并在一个独立的库中予以实现。使用这组函数需要包含此头文件，并链接该库
  - #include <dlfcn.h>
  - -ldl

知识讲解

加载共享库

- 将共享库载入内存并获得其访问句柄

void* dlopen (const char* *filename*, int *flag*);

成功返回共享库句柄，失败返回NULL

  - ***filename***：共享库路径，若只给文件名，则根据LD_LIBRARY_PATH环境变量搜索
  - ***flag***：加载方式，可取以下值

    RTLD_LAZY - 延迟加载，使用共享库中的符号(如调用库中的函数)时才加载

    RTLD_NOW - 立即加载

- 共享库访问句柄唯一地标识了系统内核所维护的共享库对象，将作为后续函数调用的参数

知识讲解

加载共享库（续1）

- 例如
  - void* handle = dlopen ("libmath.so", RTLD_NOW);
    if (! handle) {
    fprintf (stderr, "加载共享库失败！\n");
    exit (EXIT_FAILURE);
    }

知识讲解

获取函数地址

- 从指定共享库中获取与给定函数名对应的函数入口地址

void* dlsym (void* *handle*, const char* *symbol*);

成功返回函数地址，失败返回NULL

  - ***handle***：共享库访问句柄
  - ***symbol***：函数名
- 所返回的函数指针是void*类型，需要造型为实际函数指针类型才能调用

知识讲解

获取函数地址（续1）

- 例如
  - int (*add) (int, int) = (int (*) (int, int))dlsym (
    handle, "add");
    if (! add) {
    fprintf (stderr, "获取函数地址失败！\n");
    exit (EXIT_FAILURE);
    }
  - int sum = add (30, 20);

知识讲解

卸载共享库

- 从内存中卸载共享库

int dlclose (void* *handle*);

成功返回0，失败返回非零

  - ***handle***：共享库访问句柄
- 所卸载的共享库未必真的会从内存中立即消失，因为其它程序可能还需要使用该库
- 只有所有使用该库的程序都显式或隐式地卸载了该库，该库所占用的内存空间才会真正得到释放
- 无论所卸载的共享库是否真正被释放，传递给dlclose函数的句柄参数都会在该函数成功返回后立即失效

知识讲解

卸载共享库（续1）

- 例如
  - if (dlclose (handle)) {
    fprintf (stderr, "卸载共享库失败！\n");
    exit (EXIT_FAILURE);
    }

获取错误信息

- 获取在加载、使用和卸载共享库过程中所发生的错误

char* dlerror (void);

有错误则返回指向错误信息字符串的指针，否则返回NULL

- 例如
  - void* handle = dlopen ("libmath.so", RTLD_NOW);
    if (! handle) {
    fprintf (stderr, "dlopen: %s\n", dlerror ());
    exit (EXIT_FAILURE);
    }

知识讲解

Technology Tarena 达内科技

动态加载

【参见：TTS COOKBOOK 】

课堂练习

- 动态加载

Technology Tarena 达内科技

辅助工具

查看符号表

- 列出目标文件、可执行文件、静态库或共享库中的符号

```
$ nm libmath.a
```

```
calc.o:
00000000 T add
0000000d T sub
show.o:
00000000 T show
```


反汇编

- 显示二进制模块的反汇编信息

```
$ objdump -S a.out
```

```
8048514:    55                              push %ebp
8048515:    89 e5                           mov  %esp,%ebp
8048517:    83 e4 f0                        and  $0xfffffff0,%esp
804851a:    83 ec 20                        sub  $0x20,%esp
804851d:    c7 44 24 04 02 00 00    movl $0x2,0x4(%esp)
```

知识讲解

去除冗余信息

- 去除目标文件、可执行文件、静态库和共享库中的符号表、调试信息等

```
$ strip a.out
```

知识讲解

查看共享库依赖

- 查看可执行文件或共享库所依赖的共享库文件

```
$ ldd a.out
```

```
linux-gate.so.1 =>  (0xb7760000)
libmath.so (0xb775c000)
libc.so.6 => /lib/i386-linux-gnu/libc.so.6 (0xb7599000)
/lib/ld-linux.so.2 (0xb7761000)
```


配置管理共享库

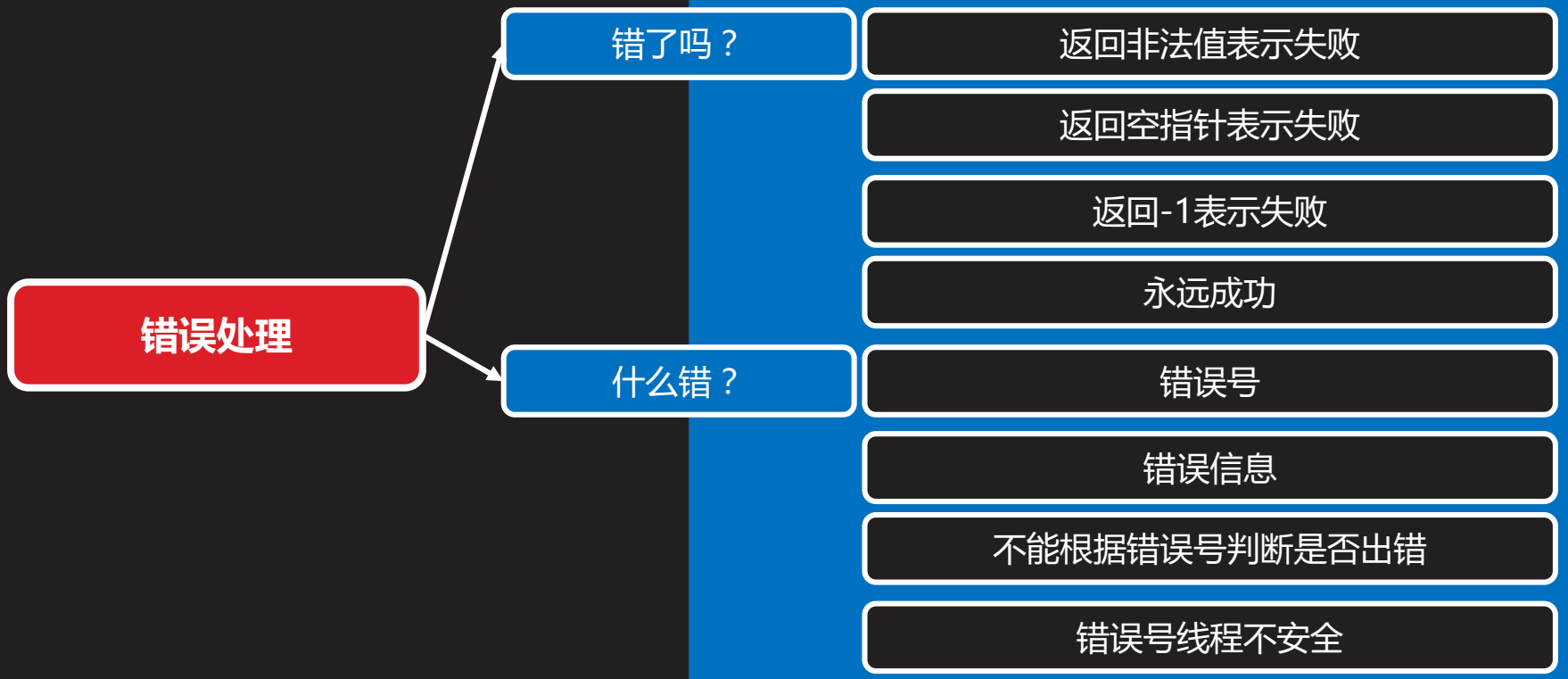
- 用专门的配置文件管理共享库的搜索路径
  - 事先将共享库的路径信息写入/etc/ld.so.conf配置文件中
  - 执行ldconfig命令，将/etc/ld.so.conf配置文件转换为/etc/ld.so.cache缓冲文件，并将后者加载到系统内存中，借以提高共享库的搜索和加载速度

```
$ ldconfig
```

  - 每次系统启动时都会自动执行ldconfig命令
  - 如果修改了共享库配置文件/etc/ld.so.conf，则需要手动执行ldconfig命令，更新缓冲文件并重新加载到系统内存

知识讲解

错误处理


错误处理
错了吗？
返回非法值表示失败
返回空指针表示失败
返回-1表示失败
永远成功
什么错？
错误号
错误信息
不能根据错误号判断是否出错
错误号线程不安全

Technology Tarena 达内科技

错了吗？

返回非法值表示失败

- 如果一个函数的返回值存在确定的合法值域，那么就可以通过返回合法值域以外的值表示函数执行失败
  - // 获取指定文件的字节数

```
long fsize (const char* path) {
    FILE* fp = fopen (path, "r");
    if (! fp)
        return -13; // 合理的文件字节数不可能为负
    fseek (fp, 0, SEEK_END);
    long size = ftell (fp);
    fclose (fp);
    return size;
}
```

知识讲解

返回非法值表示失败

【参见：TTS COOKBOOK 】

课堂练习

- 返回非法值表示失败

返回空指针表示失败

- 如果一个函数的返回值是一个指针，那么就可以通过返回空指针(即值为0的指针，通常用NULL宏表示)表示函数执行失败
  - // 求两个参数字符串中的最大值

```
const char* strmax (const char* a, const char* b) {
    return a && b ? (strcmp (a, b) > 0 ? a : b) : NULL;
}
```

知识讲解

返回空指针表示失败

【参见：TTS COOKBOOK 】

课堂练习

- 返回空指针表示失败

返回-1表示失败

- 返回0表示成功，返回-1表示失败，不输出数据或通过指针型参数输出数据
  - // 写日志

```
int plog (const char* filename, const char* log) {
    FILE* fp = fopen (filename, "a");
    if (! fp) return -1;
    fprintf (fp, "%s\n", log);
    fclose (fp); return 0; }
```

  - // 求模

```
int mod (int a, int b, int* c) {
    if (b == 0) return -1;
    *c = a % b;
    return 0; }
```

知识讲解

返回-1表示失败

【参见：TTS COOKBOOK 】

课堂练习

- 返回-1表示失败

永远成功

- 不可能失败或完全由自身处理失败的函数，无需考虑错误处理
  - // 求和
    int sum (int a, int b) {
        return a + b;
    }
  - // 释放内存
    void safe_free (void** pp) {
        if (pp) {
            free (*pp);
            *pp = NULL;
        }
    }

知识讲解

Technology Tarena 达内科技

什么错？

错误号

- 系统预定义的整型全局变量errno中存储了最近一次系统调用的错误编号
- 头文件errno.h中包含对errno全局变量的外部声明和各种错误号的宏定义
  - #include <errno.h>
  - if (errno == EEXIST) ...

知识讲解

错误信息

- 将整数形式的错误号转换为有意义的字符串

```
#include <string.h>

char* strerror (int errnum);
```

返回与参数错误号对应的描述字符串指针

- char* p = (char*)malloc (1073741824);
  if (! p) {
  fprintf (stderr, "malloc: %s\n", strerror (errno));
  exit (EXIT_FAILURE);
  }

知识讲解

错误信息（续1）

- 在标准出错上打印最近一次系统调用的错误信息

```
#include <stdio.h>

void perror (const char* s);
```

  - char* p = (char*)malloc (1073741824);
    if (! p) {
        perror ("malloc");
        exit (EXIT_FAILURE);
    }

- 用%m格式化标记打印错误信息
  - fprintf (stderr, "malloc: %m");

知识讲解

错误号和错误信息

【参见：TTS COOKBOOK 】

课堂练习

- 错误号和错误信息

不能根据错误号判断是否出错

- 虽然所有的错误号都不是零，但是因为在函数执行成功的情况下错误号全局变量errno不会被修改，所以不能用该变量的值为零或非零，做为出错或没出错的判断依据
  - char* p = (char*)malloc (0xffffffff);
    FILE* fp = fopen ("/etc/passwd", "r");
    if (errno)
        fprintf (stderr, "无法打开口令文件！\n");
- 正确的做法是，先根据函数的返回值判断是否出错，在确定出错的前提下再根据errno的值判断具体出了什么错
  - FILE* fp = fopen ("/etc/passwd", "r");
    if (! fp)
        fprintf (stderr, "无法打开口令文件：%s\n",
            strerror (errno));

知识讲解

不能根据错误号判断是否出错

【参见：TTS COOKBOOK 】

课堂练习

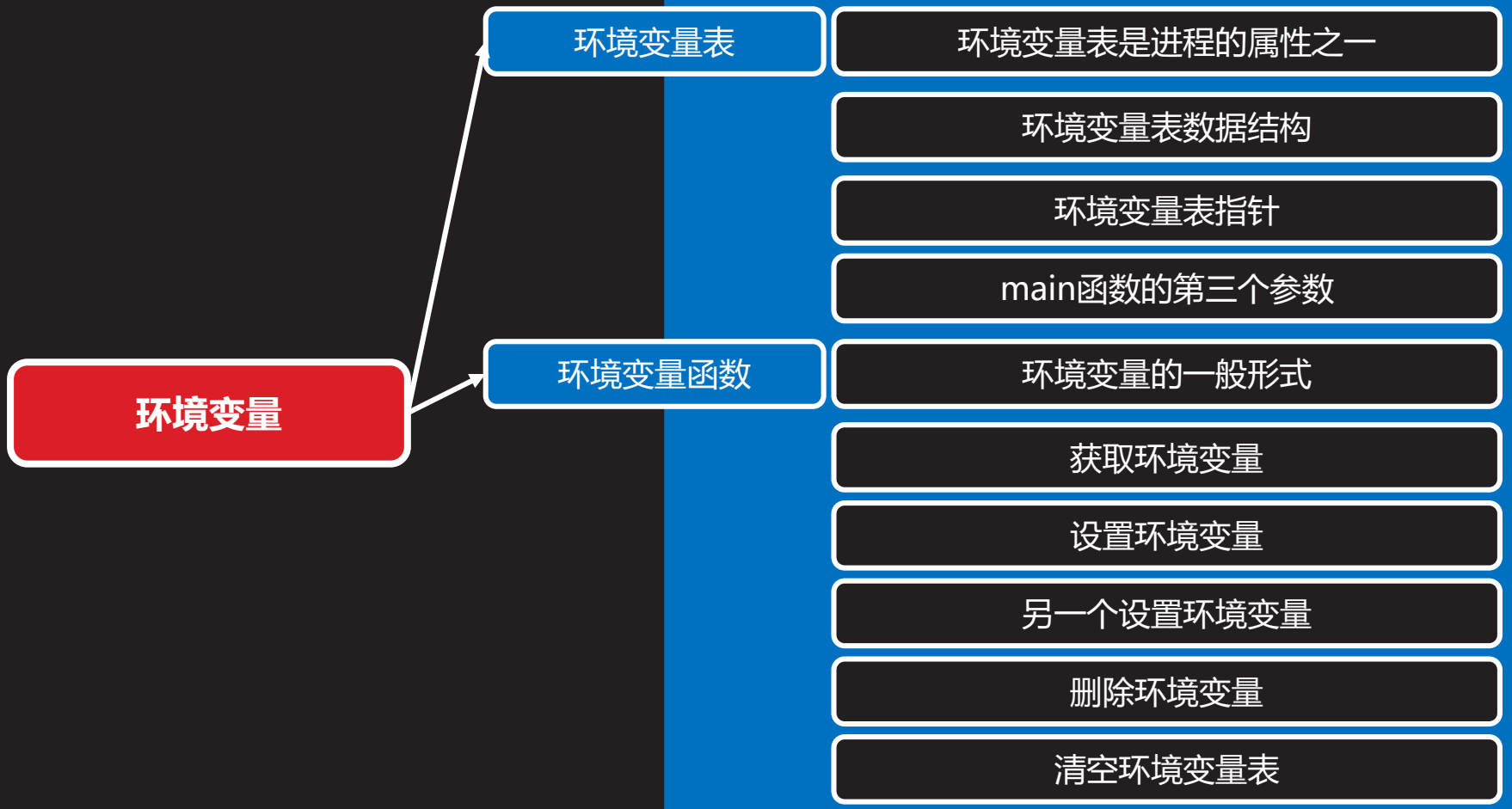
- 不能根据错误号判断是否出错

错误号线程不安全

- errno是一个全局变量，其值随时可能发生变化，尤其在多线程应用中
  - FILE* fp = fopen (
    "none", "r");
    if (! fp) {
    perror ("fopen");
    exit (EXIT_FAILURE);
    }
  - char* p = (char*)malloc (
    0xffffffff);
    if (! p) {
    perror ("malloc");
    exit (EXIT_FAILURE);
    }

知识讲解

环境变量


环境变量
环境变量表
环境变量函数
环境变量表是进程的属性之一
环境变量表数据结构
环境变量表指针
main函数的第三个参数
环境变量的一般形式
获取环境变量
设置环境变量
另一个设置环境变量
删除环境变量
清空环境变量表

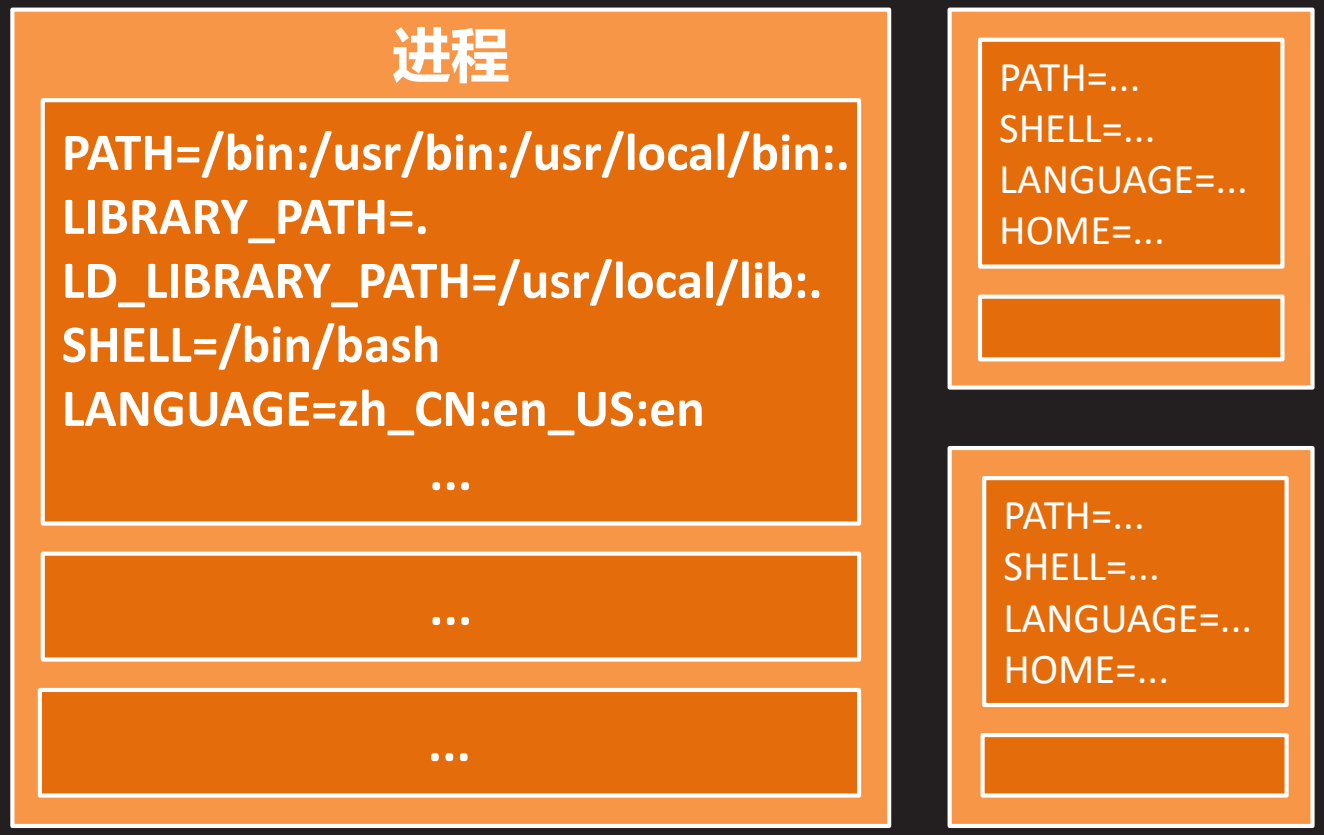
Technology Tarena 达内科技

环境变量表

环境变量表是进程的属性之一

- 每个进程都拥有一张独立的环境变量表，其中保存着专属于该进程的所有环境变量

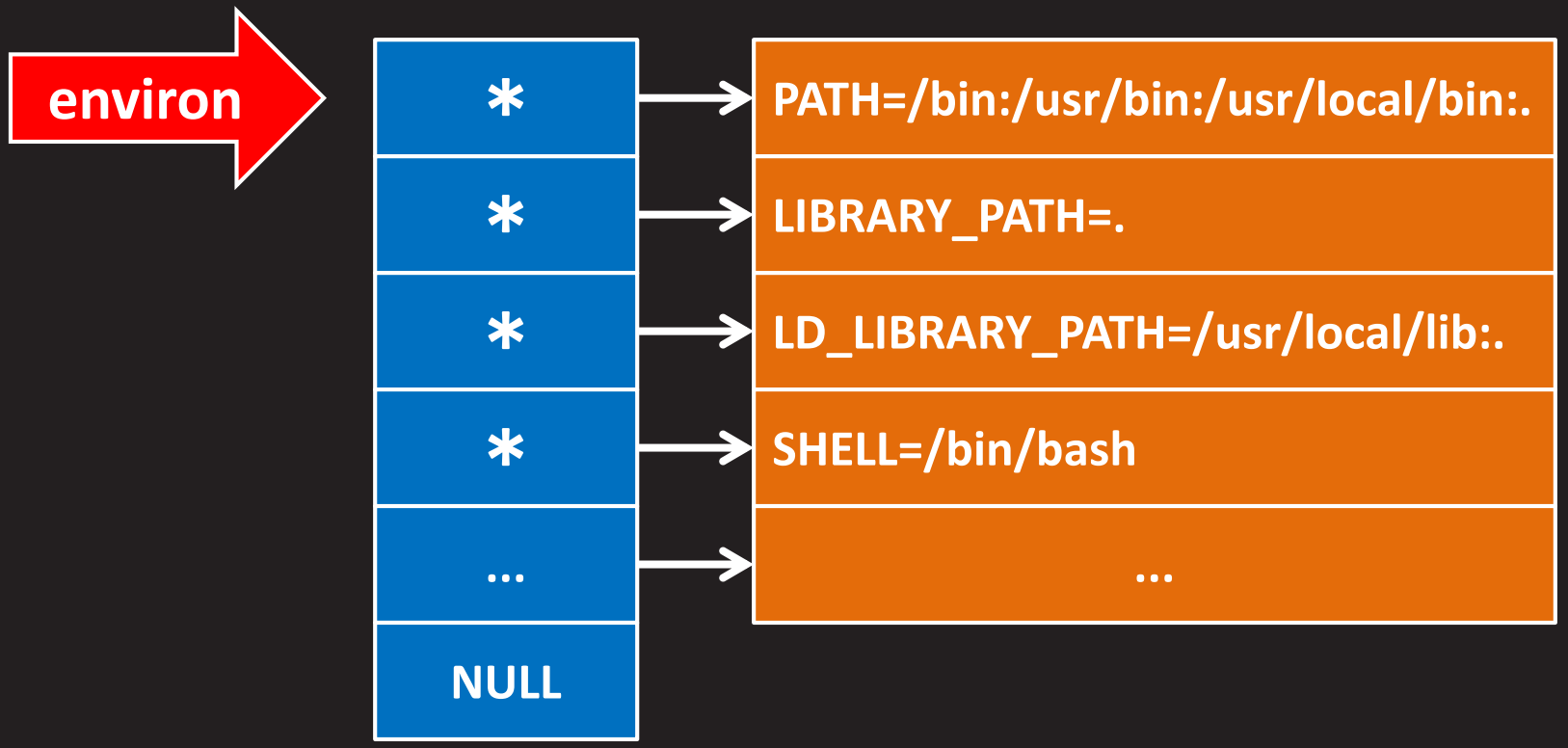
知识讲解



环境变量表数据结构

- 环境变量表是一个以空指针结尾的字符指针数组，其中每个指针指向一个格式为“变量名=变量值”的字符串，该指针数组的起始地址保存在全局变量environ中

知识讲解



环境变量表指针

- 通过全局环境变量表指针environ可以访问所有环境变量
  - extern char** environ;
    char** pp;
    for (pp = environ; pp && *pp; ++pp)
        printf ("%s\n", *pp);

知识讲解

main函数的第三个参数

- main函数的第三个参数就是环境变量表的起始地址
  - int main (int argc, char* argv[], char* envp[]) {

```
extern char** environ;
printf ("%p, %p\n", environ, envp); // 相等
char** pp;
for (pp = envp; pp && *pp; ++pp)
    printf ("%s\n", *pp);
return 0;
}
```

知识讲解

Technology Tarena 达内科技

环境变量函数

环境变量的一般形式

- 变量名=变量值
  - PATH=/bin:/usr/bin:/usr/local/bin:.
    环境变量PATH的值为/bin:/usr/bin:/usr/local/bin:.
- 进程对环境变量的操作包括
  - 根据变量名获取变量值
  - 根据变量名设置变量值
  - 增加新的环境变量
  - 删除已有的环境变量
  - 清除所有的环境变量
- 所有对环境变量的操作所影响的都仅仅是调用进程自己的环境变量，对其它进程包括其父进程，没有任何影响

知识讲解

获取环境变量

- 调用任何操作环境变量的函数都需要包含标准库头文件
  - #include <stdlib.h>
- 根据环境变量的名称获取该变量的值

char* getenv (const char* *name*);

成功返回与参数匹配的环境变量的值，失败返回NULL

  - ***name***：环境变量名
- 例如
  - printf ("PATH=%s\n", getenv ("PATH"));

知识讲解

设置环境变量

- 增加新的环境变量或修改已有环境变量的值

int putenv (char* *string*);

成功返回0，失败返回非0

  - ***string***：指向形如“变量名=变量值”的字符串，若变量名不存在就增加该环境变量，否则就修改该环境变量的值
- 例如
  - putenv ("MYNAME=zhangfei"); // 增加
    putenv ("MYNAME=zhaoyun"); // 修改

知识讲解

另一个设置环境变量

- 增加新的环境变量或修改已有环境变量的值

```
int setenv (const char* name, const char* value,
    int overwrite);
```

成功返回0，失败返回-1

  - ***name***：环境变量名，若该变量名不存在就增加该环境变量，否则由***overwrite***参数决定是否修改该环境变量的值
  - ***value***：环境变量值
  - ***overwrite***：当***name***参数所表示的环境变量已存在时，若此参数非0则将该环境变量的值修改为***value***，否则该环境变量的值保持不变

知识讲解

另一个设置环境变量（续1）

- 例如
  - setenv ("MYHOME", "Zibo", 0); // MYHOME=Zibo
    setenv ("MYHOME", "Yantai", 0); // MYHOME=Zibo
    setenv ("MYHOME", "Dezhou", 1); // MYHOME=Dezhou
- putenv的指针型参数被直接放到环境变量表中，而setenv的指针型参数则是将其目标字符串复制到环境变量表中
  - char envar[256] = "MYNAME=zhangfei";
    putenv (envar);
    strcpy (envar, "MYNAME=zhaoyun"); // 修改了环境变量
  - char name[128] = "MYHOME", value[128] = "Zibo";
    setenv (name, value, 0);
    strcpy (value, "Dezhou"); // 对环境变量毫无影响

知识讲解

删除环境变量

- 根据环境变量的名称删除环境变量

int unsetenv (const char* *name*);

成功返回0，失败返回-1

  - ***name***：环境变量名
- 例如
  - unsetenv ("MYNAME");
    unsetenv ("MYHOME");
    unsetenv ("PATH");

知识讲解

清空环境变量表

- 清除所有的环境变量

int clearenv (*void*);

成功返回0，失败返回非0

  - 该函数在清除掉所有环境变量后，把用于表示环境变量表首地址的全局变量environ设置成空指针

- 例如
  - clearenv ();
    extern char** environ;
    printf ("%p\n", environ); // (nil)

知识讲解

Technology Tarena 达内科技

环境变量

【参见：TTS COOKBOOK 】

课堂练习

- 环境变量

Technology Tarena 达内科技

总结和答疑