UC Day08

预习课

预习内容

退出处理函数

退出处理函数

退出处理函数

- 如果在程序中注册的退出处理函数，那么当程序正常执行结束之前，就会自动调用该函数
- 可以将程序的收尾性工作，放在退出处理函数中
- 退出处理函数，需要经过注册才能使用

知识讲解

退出处理函数

知识讲解

- 注册退出处理函数
  - #include <stdlib.h>
  - int atexit (void (*function) (void));

    参数：function 函数指针，指向退出处理函数

    返回值：成功返回0,失败返回-1
  - 注意atexit函数本身并不调用退出处理函数，而只是将function参数所表示的退出处理函数地址，保存(注册)在系统内核的某个地方(进程表项)。待到exit函数被调用或在main函数里执行return语句时，再由系统内核根据这些退出处理函数的地址来调用它们。此过程亦称回调

退出处理函数

- 注册退出处理函数
  - #include <stdlib.h>
  - int on_exit (void (*function) (int , void*), void* arg);

    参数：function 函数指针，指向退出处理函数。其中第一个参数来自传递给exit函数的status参数或在main函数里执行return语句的返回值，而第二个参数则来自传递给on_exit函数的arg参数

    arg 泛型指针，将作为第二个参数传递给function所指向的退出处理函数

    返回值：成功返回0,失败返回-1

知识讲解

直播课见

UC

C/C++教学体系

目录

进程的终止

正常终止

知识讲解

- 1、从main函数中返回可令进程终止
  - 进程是内存中的代码和数据，而线程则是执行代码的过程。每个进程可以包含一个多个线程，但至少要有一个主线程。每个线程都可以被看做是在一个独立的执行过程中调用了一个特殊的函数，谓之线程过程函数。线程开始，线程过程函数即被调用，线程过程函数一旦返回，线程即告终止。因此main函数也可以被看做是进程的主线程的线程过程函数。main函数一旦返回，主线程即终止，进程即终止，进程一旦终止，进程中的所有线程统统终止。这就是main函数的返回与其它函数的返回在本质上的区别。
  - main函数的返回值即进程的退出码，父进程可以在回收子进程的同时获得该退出码，以了解导致其终止的具体原因。

正常终止

知识讲解

- 2、调用exit函数令进程终止
  - #include <stdlib.h>
  - void exit(int status);

    功能：令进程终止

    参数：status　进程的退出码，相当于main函数的返回值

    该函数不返回！！！
  - 虽然exit函数的参数和main函数的返回值都是int类型，但只有其中最低数位的字节可被其父进程回收，高三个字节会被忽略，因此在设计进程的退出码时最好不要超过一字节的值域范围

正常终止

知识讲解

- 2、调用exit函数令进程终止
  - 通过return语句终止进程只能在main函数中实现，但是调用exit函数终止进程可以在包括main函数在内的任何函数中使用。
  - exit函数在终止调用进程之前还会做几件收尾工作
    - A、调用实现通过atexit或on_exit函数注册的退出处理函数；
    - B、冲刷并关闭所有仍处于打开状态的标准I/O流；
    - C、删除所有通过tmpfile函数创建的临时文件；
    - D、_exit(status);

正常终止

- 2、调用exit函数令进程终止
  - 习惯上，还经常使用EXIT_SUCCESS和EXIT_FAILURE两个宏作为调用exit函数的参数，分别表示成功和失败。它们的值在多数系统中被定义成0和-1，但一般建议使用宏，这样做兼容性更好

知识讲解

正常终止

知识讲解

- 3、调用_exit/_Exit函数令进程终止
  - #include <unistd.h>
  - void _exit(int status);

    参数：status　进程退出码，相当于main函数的返回值

    该函数不返回！
  - #include <stdlib.h>
  - void _Exit(int status);

    参数：status　进程退出码，相当于main函数的返回值

    该函数不返回！

正常终止

知识讲解

- 3、调用_exit/_Exit函数令进程终止
  - _exit在终止调用进程之前也会做几件收尾工作，但与exit函数所做的不同。事实上，exit函数在做完它那三件收尾工作之后紧接着就会调用_exit函数
    - A、关闭所有仍处于打开状态的文件描述符
    - B、将调用进程的所有子进程托付给init进程收养
    - C、向调用进程的父进程发送SIGCHLD(17)信号
    - D、令调用进程终止运行，将status的低八位作为退出码保存在其终止状态中

异常终止

知识讲解

- 1、当进程执行了某些在系统看来具有危险性的操作，或系统本身发生了某种故障或意外，内核会向相关进程发送特定的信号。如果进程无意针对收到的信号采取补救措施，那么内核将按照缺省方式将进程杀死，并视情形生成核心转储文件(core)以备事后分析，俗称吐核
  - SIGILL(4)：进程试图执行非法指令
  - SIGBUS(7)：硬件或对齐错误
  - SIGFPE(8)：浮点异常
  - SIGSEGV(11)：无效内存访问
  - SIGPWR(30)：系统供电不足

异常终止

知识讲解

- 2、人为触发信号
  - SIGINT(2)：Ctrl+C
  - SIGQUIT(3)：Ctrl+\
  - SIGKILL(9)：不能被捕获或忽略的进程终止信号
  - SIGTERM(15)：可以被捕获或忽略的进程终止信号

异常终止

知识讲解

- 3、向进程自己发送信号
  - #include <stdlib.h>
  - void abort(void);

    功能：向调用进程发送SIGABRT(6)信号，该信号默认情况下可使进程结束

    无参数，不返回!

为什么要回收子进程

为什么要回收子进程

知识讲解

- 清除僵尸进程，避免消耗系统资源。
- 父进程需要等待子进程的终止，以继续后续工作。
- 父进程需要知道子进程终止的原因
  - 如果是正常终止，那么进程的退出码是多少?
  - 如果是异常终止，那么进程是被那个信号所终止的?

wait

wait

知识讲解

- #include <sys/wait.h>
- pid_t wait(int* status);
  - 功能：等待并回收任意子进程
  - 参数：status 用于输出子进程的终止状态，可置NULL
  - 返回值：成功返回所回收的子进程的PID，失败返回-1

wait

知识讲解

- 在任何一个子进程终止前，wait函数只能阻塞调用进程，如果有一个子进程在wait函数被调用之前，已经终止并处于僵尸状态，wait函数会立即返回，并取得该子进程的终止状态，同时子进程僵尸消失。由此可见wait函数主要完成三个任务
  - 1. 阻塞父进程的运行，直到子进程终止再继续，停等同步
  - 2. 获取子进程的PID和终止状态，令父进程得知谁因何而死
  - 3. 为子进程收尸，防止大量僵尸进程耗费系统资源
- 以上三个任务中，即使前两个与具体需求无关，仅仅第三个也足以凸显wait函数的重要性，尤其是对那些多进程服务器型的应用而言

wait

知识讲解

- 子进程的终止状态通过wait函数的status参数输出给该函数调用者。<sys/wait.h>头文件提供了几个辅助分析进程终止状态的工具宏
- WIFEXITED(status)
  - 真：正常终止　　WEXITSTATUS(status) -> 进程退出码
  - 假：异常终止　　WTERMSIG(status) -> 终止进程的信号
- WIFSIGNALED(status)
  - 真：异常终止　　WTERMSIG(status) -> 终止进程的信号
  - 假：正常终止　　WEXITSTATUS(status) -> 进程退出码

waitpid

waitpid

知识讲解

- #include <sys/wait.h>
- pid_t waitpid(pid_t pid, int* status, int options);
  - 功能：等待并回收任意或特定子进程
  - 参数：pid 可以以下取值

    -1 等待并回收任意子进程，相当于wait函数

    >0 等待并回收特定子进程

    status 用于输出子进程的终止状态，可置NULL

    options 可以如下取值

    0 阻塞模式，若所等子进程仍在运行，则阻塞直至其终止。

    WNOHANG 非阻塞模式，若所等子进程仍在运行，则返回0
  - 返回值：成功返回所回收子进程的PID或者0，失败返回-1

waitpid

- 事实上，无论一个进程是正常终止还是异常终止，都会通过系统内核向其父进程发送SIGCHLD(17)信号。父进程可以忽略该信号，也可以提供一个针对该信号的信号处理函数，在信号处理函数中以异步的方式回收子进程。这样做不仅流程简单，而且僵尸的存活时间短，回收效率高

知识讲解

谢谢

UC Day08

复习课

孤儿进程

孤儿进程

- 父父进程创建子进程以后，子进程在操作系统的调度下与其父进程同时运行如果父进程先于子进程终止，子进程即成为孤儿进程，同时被某个专门的进程收养，即成为该进程的子进程，因此该进程又被称为孤儿院进程
- 接下来我们通过代码，来实际演示下孤儿进程

知识讲解

孤儿进程

- 代码思路
  - int main(void){

    //父进程创建子进程

    //子进程代码中，睡眠

    //父进程代码中，快速结束
  - }

知识讲解

下节课见