4.3 进程间通讯 - 信号（一）_物联网 / 嵌入式工程师 - 慕课网

> 慕课网慕课教程 4.3 进程间通讯 – 信号（一）涵盖海量编程基础技术教程，以图文图表的形式，把晦涩难懂的编程专业用语，以通俗易懂的方式呈现给用户。

- 信号是在软件层次上 是一种通知机制, 对中断机制的一种模拟，是一种异步通信方式, 一般具有如下特点:
    - 进程在运行过程中，随时可能被各种信号打断，
    - 进程可以忽略, 或者去调用相应的函数去处理信号
    - 进程无法预测到达的精准时间

- 在 Linux 中信号一般的来源如下:
    - 程序执行错误，如内存访问越界，数学运算除 0
    - 由其他进程发送
    - 通过控制终端发送 如 ctrl + c
    - 子进程结束时向父进程发送的 SIGCLD 信号
    - 程序中设定的定时器产生的 SIGALRM 信号

- 在 Linux 系统可以通过 kill –l 命令查看, 常用的信号列举如下:
    - SIGINT 该信号在用户键入 INTR 字符 (通常是 Ctrl–C) 时发出，终端驱动程序发送此信号并送到前台进>程中的每一个进程。
    - SIGQUIT 该信号和 SIGINT 类似，但由 QUIT 字符 (通常是 Ctrl–) 来控制。
    - SIGILL 该信号在一个进程企图执行一条非法指令时 (可执行文件本身出现错误，或者试图执行数据段、堆栈溢出时) 发出。
    - SIGFPE 该信号在发生致命的算术运算错误时发出。这里不仅包括浮点运算错误，还包括溢出及除数 > 为 0 等其它所有的算术的错误。
    - SIGKILL 该信号用来立即结束程序的运行，并且不能被阻塞、处理和忽略。
    - SIGALRM 该信号当一个定时器到时的时候发出。
    - SIGSTOP 该信号用于暂停一个进程，且不能被阻塞、处理或忽略。
    - SIGTSTP 该信号用于交互停止进程，用户可键入 SUSP 字符时 (通常是 Ctrl–Z) 发出这个信号。
    - SIGCHLD 子进程改变状态时，父进程会收到这个信号
    - SIGABRT 进程异常中止

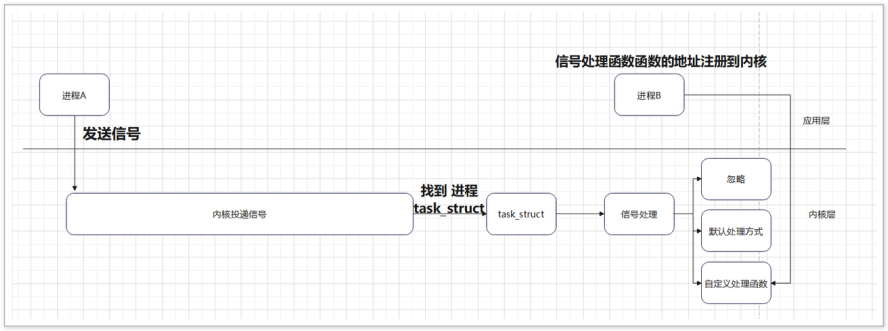
- 信号在操作系统中的定义如下:

```
#define SIGHUP       1
#define SIGINT       2
#define SIGQUIT      3
#define SIGILL       4
#define SIGTRAP      5
#define SIGABRT      6
#define SIGIOT       6
#define SIGBUS       7
#define SIGFPE       8
#define SIGKILL      9
#define SIGUSR1     10
#define SIGSEGV     11
#define SIGUSR2     12
#define SIGPIPE     13
#define SIGALRM     14
#define SIGTERM     15
#define SIGSTKFLT   16
#define SIGCHLD     17
#define SIGCONT     18
#define SIGSTOP     19
```

```
#define SIGTSTP     20
#define SIGTTIN     21
```

- 信号处理流程包含以下三个
    - 信号的发送 ： 可以由进程直接发送
    - 信号投递与处理：由内核进行投递给具体的进程并处理

- 在 Linux 中对信号的处理方式如下:
    - 忽略信号, 即对信号不做任何处理，但是有两个信号不能忽略：即 SIGKILL 及 SIGSTOP。
    - 捕捉信号, 定义信号处理函数，当信号发生时，执行相应的处理函数。
    - 执行缺省操作，Linux 对每种信号都规定了默认操作



- 在内核中的用于管理进程的结构为 task_struct , 具体定义如下:

```
struct task_struct {
    volatile long state;    /* -1 unrunnable, 0 runnable, >0 stopped */
    void *stack;
    atomic_t usage;
    unsigned int flags; /* per process flags, defined below */
    unsigned int ptrace;
```

```
    struct signal_struct *signal;
    struct sighand_struct *sighand;
```

- 当由进程来发送信号时, 则可以调用 kill() 函数与 raise () 函数

函数头文件

#include <sys/types.h>

#include <signal.h>

函数原型

int kill(pid_t pid, int sig);

- 函数功能
-
    - 向指定的进程发送一个信号
- 函数参数
-
    - pid : 进程的 id
    - sig : 信号的 id
- 函数返回值

- 成功: 返回 0
- 失败: 返回 –1, 并设置 errno

函数头文件

#include <sys/types.h>

#include <signal.h>

- 函数原型
  - int raise(int sig);
- 函数参数
  - sig : 信号编号
- 函数返回值
  - 成功 : 返回 0
  - 失败 : 返回 –1, 并设置 errno

示例 :

创建一个子进程，子进程通过信号暂停，父进程发送 终止信号

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
#include <signal.h>
#include <sys/wait.h>

int main(void)
{
    pid_t cpid;

    cpid = fork();
    if ( cpid < 0){
        perror("[ERROR]  fork() : \n");
        exit(0);
    }else if(cpid == 0){
        fprintf(stdout,"\tchild %d running.\n",getpid());

        raise(SIGSTOP);

        fprintf(stdout,"\t child %d exit \n",getpid());
        exit(EXIT_SUCCESS);

    }else if(cpid > 0){
        int status,ret;

        sleep(1);

        ret = kill(cpid,SIGKILL);

        if(ret == 0){
            fprintf(stdout,"Father %d Killed child %d\n",getpid(),cpid);
        }

        waitpid(cpid,NULL,0);

        fprintf(stdout,"father %d exit\n",getpid());

        exit(EXIT_SUCCESS);
    }
    return 0;
}
```

- 在进程没有结束时，进程在任何时间点都可以接受到信号
- 需要阻塞等待信号时，则可以调用 pause() 函数, 具体如下

函数头文件

#include <unistd.h>

函数原型

int pause(void);

函数功能

- 函数返回值
  - 成功：返回 0
  - 失败：返回 –1, 并设置 errno

示例

- 创建创建一个子进程, 父进程调用 pause 函数，子进程给父进程发送信号

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <signal.h>
#include <unistd.h>

int main(void)
{
    pid_t cpid;

    cpid = fork();

    if(cpid == -1){
       perror"[ERROR] fork(): ");
        exit(EXIT_FAILURE);
    }
    else if(cpid == 0){
        fprintf(stdout,"Child Process Start.\n");
        sleep(3);
        kill(getppid(),SIGUSR1);
        exit(EXIT_SUCCESS);
    }
    else if(cpid > 0){
        sleep(1);

        fprintf(stdout,"Main Process Start..\n");

        pause();


        fprintf(stdout,"Main Process End.\n");
    }
    return 0;
}
```

注意:

pause 函数一定要在收到信号之前调用，让进程进入到睡眠状态

练习:

创建两个子进程, 由父进程分别给两个子进程发送 SIGKILL 信号

全文完