Lab06 Assignment

> 班级：
> 学号：
> 姓名：

1. 请简述信号什么时候处于未决状态，并简述信号存在未决状态的作用

> 已经产生的信号，但是还没有传递给任何进程，此时该信号的状态就称为未决状态。
>
> 如果信号被阻塞，内核会保持信号未决，此时信号不会丢失，取消阻塞后依然可以递送信号。

2. 编写程序实现如下功能

程序 A.c 通过 sigqueue()函数按用户输入向程序 B.c 发送信号和**你的学号**作为附加数据；B.c 程序接收到该信号后，输出伴随信号的附加数据(**即你的学号**)。运行过程如下：

```
./B & # 此时，输出进程 B 的 PID 号。
./A processB_PID 19373075 # 第一个参数表示进程 B 的 PID，第二个参数为你的学号。
```

```
//code of A
#include <stdio.h>
#include <signal.h>

int main(int argc, char *argv[]){
    int message;
    pid_t B_pid;
    sscanf(argv[1],"%d",&B_pid);
    sscanf(argv[2],"%d",&message);

    union sigval info;
    info.sival_int = msg;

    sigqueue(B_pid, SIGINT, info);
    printf("A send signal with message: %d\n"info.sival_int);
    return 0;
}
```

```
//code of B
#include <stdio.h>
#include <signal.h>
#include <unistd.h>
void int_Handler(int signo, siginfo_t *info, void *ucontext){
    printf("B recieve signal with message: %d\n", info->si_value.sival_int);
}
int main(){
    printf("B pid: %d\n",getpid());
    struct sigaction act;
    act.sa_flags = SA_SIGINFO;
```

```
    act.sa_sigaction = int_Handler;
    sigaction(SIGINT, &act, NULL);
    pause();
    return 0;
}
```

3. 请实现这样一个程序

程序每间隔 1 秒输出你的学号，当按下 ctrl+c 后，程序询问是否退出程序（此时停止输出学号），输入 Y 或 5 秒未进行任何输入则退出程序，输入 N 程序恢复运行，继续输出学号（提示：alarm()函数设置超时时间，SIGALRM 信号处理函数作为超时处理）。

```
//code
#include<stdio.h>
#include<signal.h>
#include<unistd.h>
#include<stdlib.h>
#include<string.h>

void SIGALRM_handler(){
    exit(0);
}
void SIGINT_handler(){
    char c[10];
    printf("\nExit?(Y/N)\n");
    alarm(5);
    scanf("%s",c);
    alarm(0);
    if(strcmp(c,"Y")==0)
        exit(0);
}
int main(){
    signal(SIGINT,SIGINT_handler);
    signal(SIGALRM,SIGALRM_handler);
    while(1){
        printf("19373075\n");
        sleep(1);
    }
    return 0;
}//code
```

4. 请实现这样一个程序

在程序中创建一个子进程，通过信号实现父子进程交替输出，父进程输出学号，子进程输出姓名，要求父进程先输出。

```
//code
#include <string.h>
#include <signal.h>
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>

int pid;
```

```
void hand1() {
    printf("19373075\n");
    sleep(1);
    kill(pid,SIGUSR1);
}
void hand2() {
    printf("许天识\n");
    sleep(1);
    kill(getppid(),SIGUSR1);
}
int main() {
    sigset_t set;
    sigemptyset(&set);
    sigaddset(&set,SIGUSR1);
    sigprocmask(SIG_BLOCK,&set,NULL);

    pid=fork();
    if(pid){
        signal(SIGUSR1, hand1);
        sigprocmask(SIG_UNBLOCK,&set,NULL);
        while(1)pause();
    }
    else{
        signal(SIGUSR1, hand2);
        sigprocmask(SIG_UNBLOCK,&set,NULL);
        kill(getppid(),SIGUSR1);
        while(1)pause();
    }
}
```

5. 父子进程

父进程等待子进程退出通常仅需调用 wait()函数，但如果子进程未退出，父进程将会一直处于阻塞态,并通过循环不断获取子进程状态，该回收子进程的方式是对 CPU 资源的浪费。子进程终止时会自动向父进程发送 SIGCHLD 信号，请通过该特性实现这样一个程序：父进程创建 5 个子进程，每个子进程输出 PID 后以不同的状态值退出，父进程使用 SIGCHLD 信号实现异步回收子进程(非忙等)，每回收一个子进程就输出该子进程的 PID 和退出状态值，需要保证任何情况下所有子进程都能回收（提示：SIGCHLD 是不可靠信号，不支持排队，考虑两个子进程同时结束的情况）

提示：利用waitpid与sleep而不是wait，实现非忙等回收子进程。

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

int num = 0;
void child_handler()
{
    int status;
    pid_t pid;
    while ((pid = waitpid(0, &status, WNOHANG)) > 0){
```

```
        printf("Get Child %d: %d\n", pid, WEXITSTATUS(status));
        num++;
    }
}
int main()
{
    signal(SIGCHLD,child_handler);
    for (int i=1;i<=5;i++){
        if (fork() == 0){
            printf("Child %d exit: %d\n",getpid(),i);
            exit(i);
        }
    }
    while (num != 5)
        sleep(10);
    return 0;
}
```

6. 异步信号安全函数

异步信号安全函数(async-signal-safe function)是可以在信号处理函数中安全调用的函数，即一个函数在返回前被信号中断，并在信号处理函数中再次被调用，均可以得到正确结果。通常情况下，不可重入函数(non-reentrant function)都不是异步信号安全函数，都不应该在信号处理函数中调用。

1.思考: 异步信号安全/不安全函数 和 不可/可重入函数有什么关系?

> **可重入函数**：可以同时调用多次该函数，且输入是可预期的(固定输入，输出就是固定的)。
>
> 满足下列条件的函数一般是不可重入的：
>
> （1）函数体内使用了静态的数据结构；
>
> （2）函数体内调用了malloc()或者free()函数；
>
> （3）函数体内调用了标准I/O函数。
>
> （4）使用全局变量。
>
> 不可重入的函数由于使用了一些系统资源，比如全局变量区，中断向量表，标准输入输出，锁等等，所以如果全局值被修改了，那么不可重入函数的输出也可能发生变化。因此同时调用多次不可重入函数可能得到不同的输出。
>
> **异步信号安全函数**：可以在信号处理函数中安全调用的函数。
>
> 信号处理程序中应当使用异步信号安全函数。因为信号是不可预期的，当进程收到信号后，就将跳转到信号处理函数去接着执行。如果信号处理函数中使用了异步信号不安全函数，那么信号处理函数可能会修改原来进程中不应该被修改的数据(例如全局资源，锁)，这样进程从信号处理函数中返回接着执行时，可能会出现不可预料的后果。
>
> 二者的关系是：不可重入函数都不是异步信号安全函数(因为它使用了全局资源，肯定不能在信号处理函数中使用了，但即便是可重入函数，也未必能在信号处理函数中使用，只是个必要条件)。异步信号安全的都是可重入的。

2.请判断下面的函数是否是异步信号安全函数，如果是请说明理由，如果不是请给出一种可能发生问题的情况。

```
int tmp;
void swap1(int* x, int* y)
{
   tmp = *x;
   *x = *y;
   *y = tmp;
}
```

```
void swap2(int* x, int* y)
{
   int tmp;
   tmp = *x;
   *x = *y;
   *y = tmp;
}
```

> `swap1()`不是异步信号安全函数，因为使用全局变量 tmp 作为中间变量。考虑一个程序内原先tmp是用来计数的变量，值为1，突然程序收到了一个信号，在信号处理函数中使用了`swap1`，改变了tmp的值，再回到程序中时就可能发生错误。
>
> `swap2()`是异步信号安全函数，因为所有变量均为局部变量。

3.由于 printf()函数使用全局缓冲区，因此它不是异步信号安全函数。为了避免可能发生的问题，其中一个解决方法是在调用 printf()函数前阻塞所有信号，并在调用后恢复。请用上述思路补全代码，实现printf()的异步信号安全版本，无需实现格式化输出（提示：sigprocmask()函数可用于阻塞多个信号）。

```
//code
void print_safe()
{
  //TODO:阻塞所有信号
  sigset_t newset, oldset;
  sigfillset(&newset);
  sigprocmask(SIG_BLOCK, &newset, &oldset);

  printf("safe print!\n")

  //TODO:恢复所有信号
  sigprocmask(SIG_SETMASK, &oldset, NULL);
}
```