Week10 Assignment

> 班级:
> 学号:
> 姓名:

阅读教材第六章并查阅网络资料，回答以下问题。

1. 编写一个程序，实现这样的功能：搜索2~65535之间所有的素数并保存到数组中，用户输入^C信号时，程序打印出最近找到的素数。

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#include<ctype.h>
#include<math.h>
#include<unistd.h>
#include<sys/types.h>
#include<signal.h>
#define ll long long
#define MAXN 100009
#define MAXE 500009
#define MAX_INT 2147483647
#define max(a,b) a>b?a:b
#define min(a,b) a>b?b:a
int a[MAXN],n=0;
int is_ok(int i){
    int j;
    for(j=2;j*j<=i;j++){
        if(i%j==0){
            return 0;
        }
    }
    return 1;
}
void sigHandler(){
    if(n!=0){
        printf("%d\n",a[n-1]);
    }
    else{
        puts("还没找到");
    }
    signal(SIGINT,sigHandler);  //linux系统也可以去掉
}
int main(){
    int i,j;
    signal(SIGINT,sigHandler);
    for(i=2;i<=65535;i++){
        if(is_ok(i)){
            a[n++]=i;
            sleep(1);
        }
```

```
    }
    return 0;
}
```

2.简述什么是可靠信号和不可靠信号，并实验SIGINT信号是可靠的还是不可靠的。

> 早期的UNIX系统中的信号机制比较简单，是利用类似bit位来实现的，就会导致出现多个信号时发生覆盖，也就是丢失信号，因此叫作不可靠信号。后来出现了可靠信号，它通过对信号的排队来实现信号的不丢失，当出现多个信号时，那些没来得及处理的信号就会排入进程的队列，等待处理或被丢弃。
>
> 对于不可靠信号
>
> 1.进程每次处理信号后，就将对信号的响应设置为默认动作
>
> 2.信号可能丢失,如果这个信号发生多次，对后到来的这类信号不排队，仅传送该信号一次，即发生了信号丢失
>
> 但是
>
> Linux支持不可靠信号，但是对不可靠信号机制做了改进：在调用完信号处理函数后，不必重新调用该信号的安装函数（信号安装函数是在可靠机制上的实现）。因此，**Linux下的不可靠信号问题主要指的是信号可能丢失。**

实验代码：

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#include<ctype.h>
#include<math.h>
#include<unistd.h>
#include<sys/types.h>
#include<signal.h>
#define ll long long
#define MAXN 100009
#define MAXE 500009
#define MAX_INT 2147483647
int a[MAXN],n=0;
char s[MAXN];
void sigHandler(int ss){
    printf("handle %d\n",ss);
    sleep(1);
    printf("hand over\n");
    signal(SIGINT,sigHandler);
}
int main(){
    int i,j;
    signal(SIGINT,sigHandler);
    while(1){
        sleep(1);
    };
    return 0;
}
```

执行发现输入了三个^C,只产生了两个输出。

3. 在执行 ping http://www.people.com.cn 时，假设该网站是可 ping 通的，但是在输入^\时，ping 命令并没有结束而是显示 ping 的成功率，但是输入^C时，ping 程序却被退出，请解释发生这一现象的原因。

> 这可能是因为ping这个程序内部编写了对SIGINT,SIGQUIT的捕获函数，其中SIGQUIT的信号处理函数不退出，只是输出成功率，SIGINT的处理函数输出信息并退出。

4.简述什么是不可重入函数，为什么信号处理函数中要尽量避免包含不可重入函数?

> **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。

5.编写一个程序，实现这样的功能：程序每隔1秒就给自身发送一个信号，程序接收到该信号后，打印出当前的时间。

提示:

- 发送的信号可以是任何能实现功能的信号。
- 打印时间的格式不做限制，任何形式都是正确的。

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<sys/types.h>
#include<signal.h>
#include<time.h>
void sigHandler(int ss){
 time_t t;
 time(&t);
 printf("%s",ctime(&t));
}
```

```
int main(int num,char *arg[]){
    signal(SIGILL,sigHandler);
    while(1){
        sleep(1);
        raise(SIGILL);
    }
    return 0;
}
```

6.编写程序实现如下功能：程序 A.c 按照用户输入的参数定时向程序 B.c 发送信号，B.c 程序接收到该信号后，打印输出一条消息。

运行过程如下：

```
./B value& # 此时，输出进程 B 的 PID 号，value 表示要输出的参数。
./A processBPID timerVal # 第一个参数表示进程 B 的 PID，第二个参数为定时时间。
```

程序A：

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<sys/types.h>
#include<signal.h>
#define ll long long
#define MAXN 100009
int a[MAXN],n=0;
char s[MAXN];
int main(int num,char *arg[]){
    int pid,val;
    pid=atoi(arg[1]);
    val=atoi(arg[2]);
    printf("%d %d\n",pid,val);
    while(1){
        sleep(val);
        kill(pid,SIGILL);
    }
    return 0;
}
```

程序B：

```
#include<stdio.h>
#include<string.h>
#include<unistd.h>
#include<sys/types.h>
#include<signal.h>
#define MAXN 100009
int a[MAXN],n=0;
char s[MAXN];
void sigHandler(int ss){
    printf("%s\n",s);
    signal(SIGILL,sigHandler);
}
```

```c
int main(int num,char * arg[]){
    int i,j;
    strcpy(s,arg[1]);
    signal(SIGILL,sigHandler);
    printf("%d\n",getpid());
    while(1){
        sleep(1);
    };
    return 0;
}
```