CS341 #35 - Working with Signals. Case studies

> Warmup - A bit puzzle! What does the following print?

```
int i = ~ 0;   printf("%d  %x\n", i,i);
int j = (1+2+4+8) & ~2; printf("%d \n", j);
```

> Signal blocking

```
sigprocmask(int how, sigset_t * set,  sigset_t * old);
how =
  SIG_BLOCK
  SIG_UNBLOCK
  SIG_SETMASK
```

and the same for `pthread_sigmask`

e.g. What does this code do?

```
  sigemptyset( &mask);
  sigaddset( &mask, SIGALRM);

  pthread_sigmask(SIG_BLOCK, &mask, &mask2);
```

How would you then reset the signal mask?

```
pthread_sigmask(________________,____________, __________);
```

> Working with signal sets.

// Suppose `sigset_t` was just a `typedef` to a `long`. Use bitwise operations to implement the following functions.

```
void sigemptyset(long * ptr) {*ptr = 0;}

void sigfillset(long * ptr) { *ptr = _________________;}


void sigaddset(long * ptr, int sig) {
                       *ptr = *ptr | ____________________;}


void sigdelset(long * ptr, int sig) {
                       *ptr = *ptr & ____________________;}
```

> Replacing signal() with sigaction()

Portable: officially supported in multi-threaded; mask

```
int sigaction(int signum, struct sigaction *act, struct sigaction *old);

struct sigaction {
        void    (*sa_handler)(int);
        void    (*sa_sigaction)(int, siginfo_t *, void *);
        sigset_t  sa_mask;
        int       sa_flags;
};
```

```
struct sigaction sa;

sa.sa_handler = handler;
sigemptyset(&sa.sa_mask);   //Also  sigfillset
sa.sa_flags = SA_RESTART;
/* Restart functions if  interrupted by handler */

sigaction(SIGINT, &sa, NULL)
```

> Wrap a program so that it cannot be stopped with CTRL-C!

```
void main(int, char**argv) {
sigset_t mask;

___________________________

sigaddset (&mask,________________);

sigprocmask(________________, ______________, _______)

execvp(argv[1], argv+1);
...
}
```

> Synchronous checking of pending signals

```
sigset_t pending;
sigpending( &pending)

int ctrl_c = sigismember(&pending, SIGINT);
if(ctrl_c) {
  puts("Mwa Mwa Mwa");
}
```

> Pending Signals and the Process Mask across fork()

sigemptyset(&mask)
sigaddsig(&mask, SIGINT);
sigprocmask(________________, & mask, &oldmask);
Send a signal to yourself:
____
pid = fork();
sigprocmask(________________,__________________, _________________);
// Ask both child and parent to lower their mask.
if(pid ==0) puts("Child is alive");
else puts("Parent is alive!");

> Demo: Using a thread to handle signals using sigwait

sigwait blocks waiting for a signal. *Will clear the pending signal*

```
static sigset_t   signal_mask;

int main (int argc, char *argv[])
{
    pthread_t  thr_id;      /* signal handler thread ID */
    sigemptyset (&signal_mask);
    sigaddset (&signal_mask, SIGINT);
    sigaddset (&signal_mask, SIGTERM);
    pthread_sigmask (SIG_BLOCK, &signal_mask, NULL);

    pthread_create(&thr_id, NULL, signal_thread, NULL);

     /* APPLICATION CODE HERE*/
}

void *signal_thread (void *arg)
{
  while(1) {
      int       sig;
      sigwait (&signal_mask, &sig);

      switch (sig) {
      case SIGINT:     /* process SIGINT  */
        ...
        break;
      case SIGTERM:    /* process SIGTERM */
        ...
        break;
      }
   }
}
```

> Case study 01$_2$: Examples of applications using signals

Apache webserver: SIGHUP - reread config file
Java: SIGQUIT - dump heap use and thread information

> Case study 10$_2$: How to delete *everything*.... CS241 style.

Step 1. **mount** your backup disks
Step 2. Have a typo,

```
MYTEMPDIR=/home/angrave/extract/123
rm -rf $MYTEMPDIRR/
```

> Case study 2: Code Complexity Metrics.

AKA How not to write C code.

"The Camry ETCS [electronic throttle control system] code was found to have 11,000 global variables. Barr described the code as “spaghetti.” Using the Cyclomatic Complexity metric, 67 functions were rated untestable (meaning they scored more than 50). The throttle angle function scored more than 100 (unmaintainable)."

"Toyota loosely followed the widely adopted MISRA-C coding rules but Barr’s group found 80,000 rule violations. Toyota's own internal standards make use of only 11 MISRA-C rules, and five of those were violated in the actual code. MISRA-C:1998, in effect when the code was originally written, has 93 required and 34 advisory rules. Toyota nailed six of them."

Source: http://www.edn.com/design/automotive/4423428/Toyota-s-killer-firmware--Bad-design-and-its-consequences

* MISRA C is a set of software development guidelines for the C programming language developed by MISRA (Motor Industry Software Reliability Association).

* The cyclomatic complexity of a section of source code is the number of linearly independent paths through this code. For instance, if the source code contained no decision points such as IF statements or FOR loops, the complexity would be 1, since there is only a single path through the code. Two nested single-condition IFs, would produce a complexity of 4.