THE UNIX KERNEL SUPPORT OF THE SIGNALS

1. In Unix System V.3, each entry in the kernel process table slot has an array of signal flags, one for each defined in the system.

2. When a signal is generated for a process, the kernel will set the corresponding signal flag in the process table slot of the recipient process.

3. If the recipient process is asleep (waiting a child to terminate or executing *pause* API) the kernel will awaken the process by scheduling it.

4. When the recipient process runs, the kernel will check the process U-area that contains an array of signal handling specifications, where each entry of the array corresponds to a signal defined in the system.

5. The kernel will consult the array to find out how the process will react to the pending signal.

6. If the array entry contains a zero value, the process will accept the default action of the signal, and the kernel will discard it.

7. If the array entry contains a one value, the process will ignore the signal.

8. Finally, if the array entry contains any other value, it is used as the function pointer for a used defined signal hander routine.

9. The kernel will setup the process to execute the function immediately, and the process will return to its current point of execution (or to some other place if signal hander does a long jump), if the signal hander does not terminate the process.

10. If there are different signals pending on a process, the order in which they are sent to a recipient process in undefined.

11. If multiple instances of a signal are pending on a process, it is implementation – dependent on whether a single instance or multiple instances of the signal will be delivered to the process.

12. In UNIX System V.3, each signal flag in a process table slot records only whether a signal is pending, but not how many of them are present.

6.3. signal

1. All UNIX systems and ANSI – C support the signal API, which can be used to define the per-signal handing method.

2. The function prototype of the signal is:

```
#include <signal.h>
void (*signal (int signal_num, void (*handler)(int))(int);
```

signal_num is the signal identifier like SIGINT or SIGTERM defined in the <signal.h>.

handler is the function pointer of a user defined signal handler function. This function should take an integer formal argument and does not return any value.

3. Example below attempts to catch the SIGTERM, ignores the SIGINT, and accepts the default action of the SIGSEGV signal.

4. The pause API suspends the calling process until it is interrupted by a signal and the corresponding signal handler does a return:

```
#include <iostream.h>
#include <signal.h>

void catch_sig(int sig_num)
{
        signal(sig_sum, catch_sig);
        cout << "catch_sig:" << sig_num << endl;
}

int main()
{
        signal(SIGTERM, catch_sig);
        signal(SIGINT, SIG_IGN);
        signal(SIGSEGV, SIG_DFL);
        pause();                        // wait for signal interruption
}
```

5. The SIG_IGN and SIG_DFL are manifest constants defined in <signal.h>

```
#define SIG_DFL     void (*)(int)0  // Default action
#define SIG_IGN     void (*)(int)1  // Ignore the signal
```

6. The return value of signal API is the previous signal handler for the signal.

7. UNIX system V.3 and V.4 support the sigset API, which has the same prototype and similar use a signal.

```
#include <signal.h>

void (*sigset (int signal_num, void (*handler)(int))(int);
```

the sigset arguments and return value is the same as that of signal.

8. Both the functions set signal handling methods for any named signal; but, signal API is unreliable and sigset is reliable.

9. This means that when a signal is set to be caught by a signal handler via sigset, when multiple instances of the signal arrive one of them is handled while other instances are blocked. Further, the signal handler is not reset to SIG_DFT when it is invoked.

Source : http://elearningatria.files.wordpress.com/2013/10/cse-iv-unix-and-shell-programming-10cs44-notes.pdf