System Programming (Notes)

chapter # 8. Signals.

8.1 Basic Signal Concepts.

→ A signal is a software notification to a process of an event

→ A signal is generated when the event that causes the signal occurs.

→ A signal is delivered when the process takes action based that signal.

→ The lifetime of a signal is from it generation to delivery.

→ A signal that is generated but not yet delivered is said to be pending.

→ The process must be running on a processor at the time of signal delivery.

→ A process catches a signal if it executes a signal handler when the signal is deliver.

→ SIG_DFL means take default action
→ SIG-IGN means ignore a signal.
→ Neithe of these action is consider to be "catching" the signal.

→ When a process is set to 'ignore' a signal, that signal is thrown away when delevired and has no effect on the process

→ The signal mask contains a list of currently blocked signals.

→ Blocked signal are not thrown away as ignored signal are. If a pending signal is blocked, it is delivered when the process unblocks the signal.

8.2 Generating Signals

→ Every signal has a symbolic name starting with SIG.

→ Signals names are define in 'signal.h' header file.

→ The name of signals represent small integers greater than 0.

→ Every signal have their default action acept two, SIGUSR1 and SIGUSR2, which are available for user.

→ From terminal signals are generated via "kill" command. The name is kill because many signals have a default action of terminating the process.

→ Synopsis:

sends a signal to a process

```
Kill -s signal_name pid...
Kill -l [exit_status]
Kill [-signal_name] pid...
Kill [-signal_number] pid...
```

gives list of all signal names

- The signal name parameter is a symbolic name of signal by omitting the leading SIG.

(4)

→ Synopsis:

```
#include <signal.h>
int kill(pid_t pid, int sig);
```

This function is called in a program to send a signal to a process

~ pid: process ID of receiver process
~ sig: signal number/name to be sent.
*S 0, *US -1 and set errno.

- switch(pid)

case>0: send to process with that pid;
case==0; send signal to the group of caller process;
case==-1: send to all process having permissions.
case<-1: send to group id equal to |pid|.

→ Synopsis:

```
#include <signal.h>
int raise(int sig);
```

This function is used by a process to send a signal to itself.

~ sig: signal number/name
*S 0, *US ≠0, set errno.

→ The 'stty -a' command reports the charateristics of the device associated with std input, including the setting of signal-generating characters. eg; intr = ^C

→ Synopsis:

```
#include<unistd.h>
unsigned alarm (unsigned seconds);
```

This function causes a SIGALRM signal to be sent to calling process after a specified number of real seconds has elapsed. A call to alarm before the previous time expires causes alarm to be reset to new value.

~ seconds: nomber of seconds

* s number of seconds remaining on alarm befor the call reset the value, or 0 it no previous alarm was set.

* us reports no error.

8.3 Manipulating Signal Mask and Signal Sets:

→ The process "signal mask" gives the set of signals that are currently blocked. The signal mask is of type "sigset_t".

→ A process can temporarily prevent a signal from being delivered by blocking it.

→ When a process blocks a signal, the OS doesn't deleiver the signal until the process unblocks the signal.

→ When a process ignores a signal, the signal is delivered and the process handles it by Throwing it away.

→ Synopsis:

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

i) int sigaddset(sigset_t *set, int signo);
ii) int sigdelset(sigset_t *set, int signo);
iii) int sigemptyset(sigset_t *set);
iv) int sigfillset(sigset_t *set);
v) int sigismember(sigset_t *set, int signo);

Signal sets are manipulated by the five functions listed above.

i) adds signo to set
ii) removes " from "
iii) initialize sigset_t to contain no signals
iv) initialize sigset_t to contain all signals
v) reports weather a signo is in sigset_t.

~ *set ; pointer to a sigset_t.
~ signo: signal name/number.
* 0 (i-iv) & 1 if found (v)
* -1 & errno (i-iv) & 0 if not found (v).

→ Synopsis:

```
#include <signal.h>
int sigprocmask(int how, const
  sigset_t *restrict set, sigset_t
  *restrict oldset);
```

A process can examine or modify its process signal mask with this function. This function should only be used by a process with single thread. Some signal cant be blocked such as SIGSTOP and SIGKILL.

~ how: specifies the manner in which

the signal mask is to be modified, can take one of the following three values:

- SIG_BLOCK: add a collection of signals to those currently blocked.
- SIG_UNBLOCK: delet a collection of signals from those currently blocked.
- SIG_SETMASK: set the collection of signal being blocked to the specified set.

~ newset: a pointer to sigset_t to be used in the modification (if not NULL).

~ oldset: If not NULL, the old set (before modification) of the process is return in this pointer.

*s 0, *us -1 & set errno.

8.4 Catching and Ignoring Signals:

→ Synopsis:

```
#include <signal.h>
int sigaction(int signo, const
```

struct sigaction *restrict act,
struct sigaction *restrict oact);

The sigaction function allows the caller to examine or specify the action associated with a specific signal.

~ sig : signal name/number for action.
~ act : pointer to ~~sig~~ struct sigaction that specifies the action to be taken.
~ oact : pointer to struct sigaction that receives the previous action associated with the signal.
* S 0, * US -1 and set errno.

→ The structure sigaction must have atleast the following members.

struct sigaction {
- void (*sa_handler)(int);
  // SIG_DFL, SIG_IGN or pointer to user define handler function.
- sigset_t sa_mask;
  // additional signal to be blocked during execution of handler.
- int sa_flags;
  // special flags and options.

*function name is the address of function.

- void(*sigaction)(int, siginfo_t *, void *);
  //read time handler
  }

→ A single handler is an ordinary function that returns void and has one integer parameter, which is set by OS to that signal number that was delivered.

→ It is possible to have a single signal handler for many signals.

→

8.5 Waiting for Signals.

→ Busy waiting means continually using CPU cycles to test for the occurrence of an event.

→ A more efficient approach of waiting for a signal is to suspend the process until the waited-for event occurs; that way

other processes can use the CPU productively.

→ Synopsis:

```
#include <unistd.h>
int pause(void);
```

This function suspend the calling ~~function~~ thread until the delevery of a signal whose action is either to execute a user-defined handler or to terminate the process. If action is to terminate 'pause' does't return; otherwis pause returns after the signal handler.

* always return -1. It interrupted by a signal, sets errno to EINTR.

→ To wait for a particular signal by using 'pause', you must determine which signal caused 'pause' to return. This info is not available directly.

→ Synopsis:

```
#include <signal.h>
int sigsuspend(const sigset_t
        *sigmask);
```

This function sets the signal mask to the one pointed to by sigmask and suspends the process until a signal is caught by the process. This function returns when handler of the caught signal returns.

~ sigmask: pointer to mask
* alway returns -1 and set errno. The sigmask is reset to the value it had before this function was called.

→ Synopsis:

```
#include <signal.h>
int sigwait (const sigset_t
*restrict sigmask, int *restrict signo);
```

This function blocks until any of signal specified by *sigmask is pending and then removes that signal from pending signal set and unblocks. When 'sigwait' returns the removed pending signal is stored in location pointed to by 'signo'

~ sigmask: pointer to block signals
~ signo: pointer to a signal number.

*s 0
*us -1 and set errno.

→ The signal is 'sigmask' should be blocked before sigwait is called.

→ The first parameter of sigsuspend holds the new signal mask, and so the signals that are not in the set are the ones that can cause 'sigsuspend' to return;
For 'sigwait' this parameter holds the set of signals to be waited for, so the signal in the set are the ones that can cause sigwait to return.

→ Unlike sigsuspend, sigwait doesn't change the process signal mask.