CSCB09 Week 9 Notes

1. **Signals:**
    - A signal is a software generated interrupt that is sent to a process by the OS because the user pressed ctrl-c or another process wants to tell something to this process.
    - A form of Inter-Process Communication (IPC).
    - Unexpected/unpredictable asynchronous events, called **interrupts**, can happen at any time. Some interrupts are:
        - Floating point error
        - Death of a child
        - Interval timer expired (alarm clock)
        - control-C (termination request)
        - control-Z (suspend request)
    - When the kernel recognizes an event, it sends a signal to the process.
    - Normal processes may send signals, too.
    - Signals are generated by:
        - Machine interrupts
        - The program itself
        - Other programs
        - The user (You can send signals from the shell)
    - Signals do not transfer data between processes.
    - <sys/signal.h> lists the signal types.
    - Use the command **man 7 signal** to get some description of various signals.
    - Syntax: **void (*signal(int sig, void (*func)(int)))(int)**
    - **Sig:** This is the signal number to which a handling function is set. I have included a list of useful signals at the end.
    - **Func:** This is a pointer to a function. This can be a function defined by the programmer or one of the following predefined functions:
        - SIG_IGN
        - SIG_DFL

      Note: Both are defined under signal handlers.
    - You need to use **#include <signal.h>**
2. **What Signals Are Used For:**
    - When a program forks into 2 or more processes, they rarely execute independently. The processes usually require some form of synchronization which is often handled by signals.
3. **Types of Signals:**
    - **SIGINT:** Terminates the process by pressing CTRL-C (^C)
    - **SIGSTOP:** Suspends the process by pressing CTRL-Z (^Z)
    - **SIGSEGV:** Segmentation fault, usually from bad pointer usage
    - **SIGPIPE:** Writing to a pipe whose read end is closed
4. **Signal Handlers:**
    - When a C program receives a signal, control is immediately passed to a function called a **signal handler**.
    - **SIG_IGN** will ignore the signal.

- **SIG_DFL** will take the default action, which is usually to terminate the process. Every signal has a default action associated with it. The default action for a signal is the action that a script or program performs when it receives a signal.
- The signal handler can execute some C statements and exit in 3 different ways:
    - Return control to the place in the program which was executing when the signal occurred.
    - Return control to some other point in the program.
    - Terminate the program by calling exit.

5. **How a Process can deal with the Signal:**
    - Take a specified user-defined action.
    - Ignore the signal altogether and carry on processing.
    - Take the default action, which is usually to terminate the process.
6. **Signal table:**
    - For each process, Unix maintains a table of actions that should be performed for each kind of signal (see below).
    - The default action can be changed for most signal types using the sigaction() function. The exceptions are SIGKILL and SIGSTOP.

Signal	**Default Action**	**Comment**
SIGINT	Terminate	Interrupt from keyboard
SIGSEGV	Terminate/Dump core	Invalid memory reference
SIGKILL	Terminate (Cannot ignore)	Kill
SIGCHLD	Ignore	Child stopped or terminated
SIGSTOP	Stop (cannot ignore)	Stop process
SIGCONT	Resume execution at the point where the process was stopped, after first handling any pending unblocked signals	Continue if stopped

7. **Sigaction:**
    - The sigaction system call is used to change the action taken by a process on receipt of a specific signal.
    - Syntax: **int sigaction(int sig, const struct sigaction *act, struct sigaction *oldact);**
    - sig specifies the signal and can be any valid signal except SIGKILL and SIGSTOP.
    - If act is non-null, the new action for signal sig is installed from act.
    - If oldact is non-null, the previous action is saved in oldact.

- The sigaction structure is defined as something like:

```
struct sigaction {
        void (*sa_handler)(int);
        // SIG_DFL, SIG_IGN, or pointer to function.
        // Pointer to a signal-catching function or one of the
          macros SIG_IGN or SIG_DFL.

        sigset_t sa_mask;
        // Signals to block during handler.
        // Additional set of signals to be blocked during
          execution of a signal-catching function.

        int sa_flags;
        // flags and options, SA_RESTART, SA_RESETHAND
        // Special flags to affect the behavior of signals.
        };
```

- **Note:** There are more extensions for the struct.
- If a signal is received when a signal handler is running, the first signal handler is suspended and the newly invoked handler runs, then the first handler is resumed.
- sa_mask can be used to block out signals while one handler is running.

8. **Blocking Signals:**
   - Signals can arrive at any time.
   - To temporarily prevent a signal from being delivered, we block it.
   - The signal is held until the process unblocks the signal, then it is delivered.
   - When a process ignores a signal, it is thrown away.
   - Blocked signals are held until they are unblocked, this is different than being ignored.
9. **Signal Set:**
   - Signal sets are used to store the set of signals that are currently blocked or unblocked.
   - Operations on signal sets:
     - **int sigemptyset(sigset_t *set);**
     - **int sigfillset(sigset_t *set);**
     - **int sigaddset(sigset_t *set, int signo);**
     - **int sigdelset(sigset_t *set, int signo);**
     - **int sigismember(const sigset_t *set, int signo);**
   - Example:
     - **sigset_t mask;**
       sigfillset(&mask) // Blocks every signal during handler
       sigemptyset(&mask) // Don't block any signals during the handler
       sigaddset(&mask, SIGQUIT) // Block SIGQUIT
       sigdelset(&mask, SIGALRM) // Unblock SIGALRM
       sigismember(&mask, SIGHUP) // Check if SIGHUP is waiting

10. **Sigprocmask:**
    - Used to fetch and/or change the signal mask of the calling thread. I.e. It examines and changes blocked signals.
    - Syntax: **int sigprocmask(int how, const sigset_t *set, sigset_t *oset);**
    - how indicates how the signal will be modified:
        - SIG_BLOCK: Add to those currently blocked
        - SIG_UNBLOCK: Delete from those currently blocked
        - SIG_SETMASK: Set the collection of signals being blocked
    - set points to the set of signals to be used for modifying the mask.
    - oset on return holds the set of signals that were blocked before the call.
11. **Sending a signal:**
    - You can use the kill command to send a signal to a process.
    - Syntax: **kill [-signal] pid [pid]…**
    - If you don't specify which signal to send, by default the **TERM** signal is sent, which terminates the process.
    - The signal can be specified by the number or the name without the SIG.
    - E.g.
        - kill -QUIT 8883
        - kill -STOP 78911
        - kill -9 76433 This is the same as kill -KILL 76433 because 9 == KILL.
12. **Signalling between processes:**
    - One process can send a signal to another process using the kill function.
    - **Note:** The kill function is not the same as the kill command.
    - Syntax: **int kill(pid_t pid, int sig);**
    - **pid:** id of destination process
    - **signal:** the type of signal to send
    - **Return value:** 0 if signal was sent successfully and -1 if the signal was not sent successfully.
    - The kill function sends a signal specified by sig to a process or a group of processes specified by pid.
    - Need to use the following:
        - **#include <sys/types.h>**
        - **#include <signal.h>**
    - Signalling between processes can be used for many purposes:
        - Kill errant processes
        - Temporarily suspend execution of a process
        - Make a process aware of the passage of time
        - Synchronize the actions of processes.
13. **Timer signals:**
    - Three interval timers are maintained for each process:
        - **SIGALRM:** Real-time alarm, like a stopwatch
        - **SIGVTALRM:** Virtual-time alarm, measuring CPU time
        - **SIGPROF:** Used for profilers

- Useful functions to set and get timer info:
    - **sleep():** Suspends the calling process
    - **usleep():** Like sleep() but at a finer granularity
    - **alarm():** Sets SIGALRM
    - **pause():** Suspends the process until next signal arrives
    - **setitimer()**
    - **getitimer()**
- sleep() and usleep() are interruptible by other signals.

14. Limitations:

- Signals don't contain data, all you know is the signal type.
- Multiple signals:
    - If repeated signals are sent before a process has a chance to check for its signals, the effect is the same as being sent only once (second signal is lost).
    - If multiple (different) signals are sent, all will be received when the process has a chance to check.
- Signals are usually used to indicate conditions and synchronization.

15. List of Signals:

- **Note:** You can use kill -l to get this.

Number	Signal	What it does
1	SIGHUP	Used to report the termination of the controlling process on a terminal to jobs associated with that session. This termination effectively disconnects all processes in the session from the controlling terminal. (Ctrl + D)
2	SIGINT	Issued if the user sends an interrupt signal (Ctrl + C)
3	SIGQUIT	Issued if the user sends a quit signal (Ctrl + \)
8	SIGFPE	Reports a fatal arithmetic error. Although the name is derived from “floating-point exception”, this signal actually covers all arithmetic errors, including division by zero and overflow.
9	SIGKILL	If a process gets this signal it must quit immediately and will not perform any clean-up operations. Cannot be blocked or ignored.
11	SIGSEGV	Generated when a program tries to read or write outside the memory that is allocated for it, or to write memory that can only be read.
13	SIGPIPE	Writing to a pipe whose read end is closed.
14	SIGALRM	Alarm clock signal (Used for timers)
15	SIGTERM	Terminates the process. (Sent by the kill command by default)
17	SIGCHLD	Sent to a parent process whenever one of its child processes terminates or stops.
18	SIGCONT	Continues a process.
19	SIGSTOP	Stops the process. (Ctrl + Z) It cannot be handled, ignored, or blocked.
26	SIGVTALRM	Virtual-time alarm, measuring CPU time.
27	SIGPROF	Indicates expiration of a timer that measures both CPU time used by the current process, and CPU time expended on behalf of the process by the system. This is used to implement code profiling facilities.