**UNIX System Programming**

**Lecture 12: Signals**

- **Outline**
  - Waiting on a Child
  - Sending Signals to a Process

- **Reference**
  - BLP: Chapter 11
  - man pages: wait, waitpid, kill, sigaction, sigsetops, pause, alarm, sleep

---

**Lecture 12: Signals Using wait()**

- **wait()** causes the parent to ________ until any child terminates:
  ```c
  int status;
  switch (fork()) {
  case 0:
    execvp(program, args);
  default:
    pid = wait(&status);
  }
  ```
- It is not necessary for a parent to wait.
- See `wait_xmpl.cpp` for example code.

---

**Lecture 12: Signals Using waitpid()**

- **waitpid()** allows you to: (1) wait on a specific process, (2) check status without ________, and (3) supports job control.
  ```c
  pid_t waitpid(pid_t pid, 
  int *status, 
  int opts)
  ```
- If `pid == -1`, `waitpid()` waits for any child. If `pid > 0`, `waitpid()` waits for child with that pid. status is defined just as for `wait()`. If opts is WNOHANG, check status and return

---

**Lecture 12: Signals Avoiding Zombies**

- If we want to break ties with (disown a) child process we can call ________ twice:

  ```c
  if ( (pid = fork()) == 0) {    // first child
    if ( (pid = fork()) == 0) {  // second child
      // our parent becomes init when first child exits
      execvp(program, args);
    }                        // first child, so exit. Second process is adopted by init
    exit(0);
  }
  ```
- `waitpid(pid, NULL, 0): // wait for first child`
  - we’re the parent — go on and do our own thing
  - we don’t have to worry about the second child

---

**Lecture 12: Signals Introduction to Signals**

- Signals are software ________. Signals are “raised” or “sent” by the kernel or a process to a process which “receives” or “catches” it.
- Signals can be raised by the kernel in response to an exception (segment violation, floating point error, illegal instruction).
- The terminal driver sends signals in response to certain key presses (CTRL-C, CTRL-Z).
- They can be used as a primitive form of IPC.
Lecture 12: Signals
Raising/Sending Signals

- The `kill()` routine is used to _____ a signal:
  ```
  #include <sys/types.h>
  #include <signal.h>
  int kill(pid_t pid, int sig);
  ```
- There is also a `kill` utility that can be used to send signals from the command line.
- `raise(int sig)` is `kill(getpid(), sig)` and can be used by a process to signal itself.

Lecture 12: Signals
Standard Signals

- Each signal has a unique name (and _____, but use the name). A complete list can be generated by entering "man 7 signal" or "kill -l" at a prompt. Here is a partial list:
  ```
  SIGHUP  Hangup
  SIGQUIT Quit
  SIGKILL Kill
  SIGCLD Child term
  SIGSTOP Stop
  SIGUSR1 User sig 1
  SIGUSR2 User sig 2
  SIGINT  Interrupt
  SIGTERM Soft term
  SIGALRM Alarm
  SIGFPE FP except.
  SIGCONT Continue
  ```

Lecture 12: Signals
Receiving Signals

- When a process receives a signal it can: (1) ignore the signal (except for SIGKILL and SIGSTOP), (2) _____ (or handle) the signal, (3) take the default action (for most signals the default is to terminate).
- To catch the signal we must install a signal (or interrupt) handler. The `signal()` routine has traditionally been used to do this, but `signal()` varies across UNIX systems.

Lecture 12: Signals
Receiving Signals

- `signum` specifies the signal and can be any valid signal except ________ and SIGSTOP.
- If `act` is non-null, the new action is installed from `act`. If `oldact` is non-null, the previous action is saved in `oldact`.
- The `sigaction` structure looks like:
  ```
  struct sigaction {
    void (*sa_handler)(int);
    sigset_t sa_mask;
    int sa_flags;
  }
  ```

Lecture 12: Signals
Receiving Signals

- `sa_handler` is a pointer to the desired signal handler function or it may be either `SIG_DFL` to restore the default action or `SIG_IGN` to __________ the signal.
- `sa_mask` is a signal set (see `sigsetops` on a later slide) that indicates which signals should be blocked during execution of the signal handler.
Lecture 12: Signals Receiving Signals

- **sa_flags** modifies the behavior of the signal handling process. It is formed by bitwise several flags. A few of which are:
  - SA_NOCLOSTOP: Block child stop notification.
  - SA_NOCLOWAIT: When SIGCHLD is received, do not transform children into zombies.
  - SA_RESETHAND: Restore the default handler after the current handler is called once.
  - SA_NODEREFER: Do not block a signal in its own signal handler.

- You can use the **pause()** routine to ______ until a signal is received. The **wait()** routine we used earlier is similar, it sleeps until SIGCHLD is received (and also gives access to the exit status).
- See **sig_xmpl.cpp** for example code that illustrates installing a new signal handler for the INT signal (CTRL-C).

Lecture 12: Signals Receiving Signals

- The **sigsetops** routines are used to ______ and manipulate signal sets (like **sa_mask**):
  int sigemptyset(p_set);
  int sigfillset(p_set);
  int sigaddset(p_set, signum);
  int sigdelset(p_set, signum);
  int sigismember(p_set, signum);

  where p_set is a “sigset_t *” type and signum is a signal number.

- **alarm(sec)** arranges for the current process to receive a ______ in **sec** seconds.
  alarm(5); // Alarm in 5 seconds
  alarm(0); // Cancel alarm

- **SIGALRM** will terminate a process by default. A signal handler is normally installed before setting the alarm.

Lecture 12: Signals Receiving Signals

- **alarm()** can be used to ___________ an operation (such as a read from a terminal):
  alarm(10); // Alarm in 10 secs
  n = read(0, line, MAXLINE);
  alarm(0); // Reset alarm

- See **alm_xmpl1.cpp** and **alm_xmpl2.cpp**
- See the **sleep(), usleep(), nanosleep()** and **getitimer()*/setitimer()** man pages for additional information.

- In addition to all of the standard signals SIGUSR1 and SIGUSR2 are ______ signals. They can be used for simple IPC.
- Modify **exercise.cpp** so that the child process (after sleeping for 3 seconds) sends the SIGUSR1 signal to the parent and exits. Modify the code for the parent process to install the signal handler. You will also need to write the signal handler code.