Lecture 9: Unix Signals (Contd) (Feb 15, 2005)
Yap

February 17, 2005

1 ADMIN

- Homework 3 out today – please begin the programming part immediately as it involves many new concepts.
- Today, we continue discussion of programming techniques needed in interprocess communication in Unix.

2 Review

- Q: What do most system calls return in case of error?

  A: Minus 1 (-1).
- Q: Write a C program that (1) creates a pipe (2) writes into the pipe a pair of numbers 123, 456 (3) reads from the pipe the 2 numbers (4) verify that that the numbers you read are 123, 456

  A:

```
//=========================================
// Writing and Reading a pair of numbers into a pipe
//
#include <stdio.h>
#include <unistd.h>

int
main()
{
        char mess[1024];
        int pv[2];
// create pipe:
        if (pipe(pv)<0)
                perror("open pipe");
```

```
// write 2 pipe
        int m=123, n=456;
        char buf[1024];
        sprintf(buf, "%d %d", m, n);
        int len=strlen(buf);
        if (write(pv[1],buf, len)< len)
                perror("write");

// read from pipe
        int mm, nn;
        char outbuf[1024];
        if (read(pv[0], outbuf, len)< len)
                perror("read");
        sscanf(outbuf,"%d %d", &mm, &nn);
        if ((m!=mm) || (n!=nn))
                printf("mm=%d, nn=%d\n", mm, nn);
        else
                printf("m=%d, n=%d\n", m, n);
        exit(0);
}
//=========================================
```

- Q: A pipe is essentially a one-way communication. How can you create a 2-way communication between a parent and a child processes?

  A: Create two pipes (and agree on which pipe should go from which child to which child).

- Q: Give the mechanics of signaling

  A: Each process sets up a disposition towards each signal (or accepts the default). The disposition besides default is to ignore or to set up a handler.

  Then it goes about doing its task and if a signal is received, the action will be taken automatically (process interrupted).

3 Signals

- Q: How do you signal a process?

  A: Use the "kill" command.

  kill(pid_t id, int signo);

  If the id is 0, then all the processes associated with the current job will get the signal.

  You can send a signal to yourself using the "raise" command.

- When

4 Non Local Branches (long jumps)

- Goto statements are much maligned, but it has its uses.
- Consider the following program:

```
\\==========================
a() {
   x:
}
b() {
   goto x;
}
main() {
  b();
}
\\==========================
```

  Where should a() return to after the goto? The semantics is totally unclear.

- Next consider:

```
\\==========================
a() {
   goto x;
   ...
   goto y;
}
b() {
   a();
   x:
}
main() {
  b();
  y:
}
\\==========================
```

  These goto's offer quick exits, and are meaningful since we always jump to active blocks.

  Implicitly, each goto must be followed by deallocation of local storage of the intervening activation blocks.

- But since we (the compiler) could not distinguish between the above two programs (one meaningless, one meaningful), we disallow such goto's.

- Instead we provide two library functions, `setjmp()` and `longjmp()`.

```
\\============================
#include <setjmp.h>

int setjmp(jmp_buf env);

void longjmp(jmp_buf env, int val);

\\============================
```

- Setjmp() saves the stack environment into a structure of type `jmp_buf`.

  When we call Longjmp(), we can use the same `jmp_buf` to restore the environment (at the point where setjmp() was first called.

- Thus, setjmp() can return in two ways: the original setting up of the environment, and all subsequent longjmp() back to this location. To distinguish them, the original setting up returns 0. The longjump() call can specify the variable (`val`) to be returned (normally, `val` should not be 0).

  THUS longjmp() does not return to its location of call. It is like exec() in this sense.

- Below, we actually use `sigsetjmp/siglongjmp` instead of `setjmp/longjmp`.

  These sig-variations take an additional integer argument. When non-zero, the current signal mask is stored in the environment (`env`). A subsequent `siglongjmp()` will restore the signal as well.

- Here is how we can use setjmp/longjmp with signals.

- ```
  //==========================================
  // file signal.c
  #include <stdio.h>
  #include <signal.h>
  #include <setjmp.h>

  jmp_buf env;


  void int_handler(int sig)
  {
          //sigset(SIGINT, int_handler);
          signal(SIGINT, int_handler);
          printf("SIGINT signal=%d\n",sig);
          siglongjmp(env,1);
  }
  ```
  ```

```
void quit_handler(int sig)
{
        signal(SIGQUIT, int_handler);
        printf("SIGQUIT signal=%d\n",sig);
        siglongjmp(env,1);
}

main()
{
        int i, k = 0;
        char buf[BUFSIZ];

        signal(SIGINT, int_handler);
        printf("SIGINT=%d\n",SIGINT);

        if (sigsetjmp(env,1) != 0){
                printf("restarted...\n");
                k=0;
        }

        for (i=0; ((i<20)&&(k<20)) ; i++) {//main loop
                printf("%d> ", ++k);
                gets(buf);
                printf("%s\n", buf);
        }
}
//===========================================
```