Nonblocking I/O and I/O multiplexing

Nonblocking I/O

- Two ways to make “slow” systems calls nonblocking:
  1. call `open()` with `O_NONBLOCK`
  2. call `fcntl()` to turn on `O_NONBLOCK` file status flag (recall that file status flag is part of file table entry – the middle layer)
- Nonblocking slow system call returns –1 with `errno` set to `EAGAIN` if it would have blocked
- Nonblocking write example:

```
#include "apue.h"
#include <errno.h>
#include <fcntl.h>

char    buf[500000];

int
main(void)
{
    int     ntowrite, nwrite;
    char    *ptr;

    ntowrite = read(STDIN_FILENO, buf, sizeof(buf));
    fprintf(stderr, "read %d bytes\n", ntowrite);

    set_fl(STDOUT_FILENO, O_NONBLOCK);  /* set nonblocking */

    ptr = buf;
    while (ntowrite > 0) {
        errno = 0;
        nwrite = write(STDOUT_FILENO, ptr, ntowrite);
        fprintf(stderr, "nwrite = %d, errno = %d\n", nwrite, errno);

        if (nwrite > 0) {
            ptr += nwrite;
            ntowrite -= nwrite;
        }
    }

    clr_fl(STDOUT_FILENO, O_NONBLOCK);  /* clear nonblocking */

    exit(0);
}
```

- `set_fl()` and `clr_fl()`:

```
#include "apue.h"
#include <fcntl.h>

void set_fl(int fd, int flags) /* flags are file status flags to turn on */
{
        int     val;

        if ((val = fcntl(fd, F_GETFL, 0)) < 0)
                err_sys("fcntl F_GETFL error");

        val |= flags;       /* turn on flags */

        if (fcntl(fd, F_SETFL, val) < 0)
                err_sys("fcntl F_SETFL error");
}

void clr_fl(int fd, int flags) /* flags are file status flags to turn off */
{
        int     val;

        if ((val = fcntl(fd, F_GETFL, 0)) < 0)
                err_sys("fcntl F_GETFL error");

        val &= ~flags;      /* turn flags off */

        if (fcntl(fd, F_SETFL, val) < 0)
                err_sys("fcntl F_SETFL error");
}
```

I/O multiplexing

- `cat`:

```
while ((n = read(STDIN_FILENO, buf, BUFSIZ)) > 0)
    if (write(STDOUT_FILENO, buf, n) != n)
        err_sys("write error");
```

- What about netcat (`nc`)?
- `select()`:

```
#include <sys/select.h>

int select(int maxfdp1, fd_set *restrict readfds,
           fd_set *restrict writefds, fd_set *restrict exceptfds,
           struct timeval *restrict tvptr);

        Returns: count of ready descriptors, 0 on timeout,–1 on error

int FD_ISSET(int fd, fd_set *fdset);

        Returns: nonzero if fd is in set, 0 otherwise

void FD_CLR(int fd, fd_set *fdset);
void FD_SET(int fd, fd_set *fdset);
void FD_ZERO(fd_set *fdset);
```

- `select()` will always get interrupted on signals, even when the `SA_RESTART` flag was used
- Nonblocking write example, augmented with `select()`:

```
#include "apue.h"
#include <errno.h>
#include <fcntl.h>
#include <sys/select.h>

char    buf[500000];

int
main(void)
{
        int     ntowrite, nwrite;
        char    *ptr;

        ntowrite = read(STDIN_FILENO, buf, sizeof(buf));
        fprintf(stderr, "read %d bytes\n", ntowrite);

        set_fl(STDOUT_FILENO, O_NONBLOCK);  /* set nonblocking */

        ptr = buf;
        while (ntowrite > 0) {
                errno = 0;

                // Wait until stdout is ready before we call write()
                fd_set write_fds;
                FD_ZERO(&write_fds);
                FD_SET(STDOUT_FILENO, &write_fds);
                select(STDOUT_FILENO + 1, NULL, &write_fds, NULL, NULL);

                nwrite = write(STDOUT_FILENO, ptr, ntowrite);
                fprintf(stderr, "nwrite = %d, errno = %d\n", nwrite, errno);

                if (nwrite > 0) {
                        ptr += nwrite;
                        ntowrite -= nwrite;
                }
        }

        clr_fl(STDOUT_FILENO, O_NONBLOCK);  /* clear nonblocking */

        exit(0);
}
```

Last updated: 2014–02–20