Non-blocking Read from a Pipe: An Example

Note: This example, taken from pages 161–162 of the text by Haviland et al., shows how `fcntl` system call can be used to change the status of the read-end of a pipe to be non-blocking.

The parent process creates the pipe and uses `fcntl` to make the read-end of the pipe non-blocking. The parent then waits in a loop, checking the pipe after sleeping for 1 second at a time. The child sends three messages to the pipe, with a two second gap between messages. When this program is executed, the output will have two "Pipe is empty" messages from the parent for each message received from the child.

```c
#define MSGSIZE 6

#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <fcntl.h>
#include <unistd.h>

int parent (int []); int child (int []); /* Prototypes. */
char *msg1 = "hello"; char *msg2 = "bye!!";

int main(void) {
    int pfd[2]; /* Pipe’s file descriptors. */
    /* Set up pipe. */
    if (pipe(pfd) == -1) {
        fprintf(stderr, "Call to pipe failed.\n"); exit(1);
    }

    /* Set O_NONBLOCK flag for the read end (pfd[0]) of the pipe. */
    if (fcntl(pfd[0], F_SETFL, O_NONBLOCK) == -1) {
        fprintf(stderr, "Call to fcntl failed.\n"); exit(1);
    }

    /* Fork a child process. */
    switch (fork()) {
    case (pid_t) -1: /* Fork failed. */
        fprintf(stderr, "Call to fork failed.\n"); exit(1);
    case 0: /* Child process. */
        child(pfd);
    default: /* Parent process. */
        parent(pfd);
    } /* End of switch. */
    return 0;
} /* End of main. */
```

(over)
int parent (int p[]) {
    /* Code for parent process. */
    #define PSLEEP_TIME 1
    int nread; char buf[MSGSIZE];

    /* Close the write-end of the pipe. */
    if (close(p[1]) == -1) { /* Failed to close write end of pipe. */
        fprintf(stderr, "Parent: Couldn't close write end of pipe.\n"); exit(1);
    }

    /* Repeatedly monitor the pipe for messages. Stop when the pipe is closed. */
    for (;;) {
        switch (nread = read(p[0], buf, MSGSIZE)) {
        case -1: /* Make sure that pipe is empty. */
            if (errno == EAGAIN) {
                printf("Parent: Pipe is empty\n"); fflush(stdout);
                sleep(PSLEEP_TIME); break;
            } else { /* Reading from pipe failed. */
                fprintf(stderr, "Parent: Couldn't read from pipe.\n"); exit(1);
            }
        case 0: /* Pipe has been closed. */
            printf("Parent: End of conversation.\n"); fflush(stdout); exit(0);
        default: /* Received a message from the pipe. */
            printf("Parent: Message -- %s\n", buf); fflush(stdout); break;
        } /* End of switch. */
    } /* End of for loop. */
} /* End of parent. */

int child (int p[]) {
    /* Code for the child process. */
    #define CSLEEP_TIME 2
    #define NUM_MSG 3
    int count;
    /* Close the read-end of the pipe. */
    if (close(p[0]) == -1) { /* Failed to close read end of pipe. */
        fprintf(stderr, "Child: Couldn't close read end of pipe.\n"); exit(1);
    }

    /* Send messages through the pipe. */
    for (count = 0; count < NUM_MSG; count++) {
        write(p[1], msg1, MSGSIZE); sleep(CSLEEP_TIME);
    }

    /* Send the final message. */
    write(p[1], msg2, MSGSIZE); exit(0);
} /* End of child. */