Jun 06, 04 12:42 **eg16−pipe.c** Page 1/1

```c
/*
 * Introducing pipe − create two file descriptors,
 * fd[0] for reading and fd[1] for writing.
 */
#include <unistd.h>

int main()
{
    int fd[2];
    char *str = "Hello World\n";
    char buf[14];

    if (pipe(fd) == −1)
    {
        perror("pipe");
    }

    write(fd[1], str, strlen(str));
    read(fd[0], buf, 12);
    buf[12] = 0;
    printf("%s", buf);

    close(fd[0]);
    close(fd[1]);
}
```

Jun 06, 04 13:17 **eg17−pipe.c** Page

```c
/*
 * pipe is a bit more useful when we fork() −
 * we can use it to communicate between parent
 * to child.
 *
 * From now on, error handling code will be
 * ommitted for simplicity (but, they should
 * be there when we write real code!)
 */
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>

int main()
{
    int fd[2];
    pid_t pid;
    char *str = "Hello World\n";
    char buf[14];

    pipe(fd);
    pid = fork();
    switch (pid) {
        case 0:
            read(fd[0], buf, 13);
            printf("child %d, read %s", getpid(), buf);
            _exit(0);
            break;
        default:
            write(fd[1], str, 13);
            printf("parent %d, write %s", getpid(), str);
    }
    return 0;
}
```

Jun 06, 04 13:21 **eg18−pipe.c** Page 1/1

```c
/*
 * We can use dup2 to pipe from stdin to stdout.
 */
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>

int main()
{
    int fd[2];
    int i, sum = 0;
    pid_t pid;
    char buf[11];

    pipe(fd);
    pid = fork();
    switch (pid) {
        case 0:
            dup2(fd[0], STDIN_FILENO);
            for (i = 0; i < 10; i++) {
                fgets(buf, 10, stdin);
                sum += atoi(buf);
            }
            printf("%d\n", sum);
            _exit(0);
            break;
        default:
            dup2(fd[1], STDOUT_FILENO);
            for (i = 0; i < 10; i++)
                printf("%d\n", i);
    }
    return 0;
}
```

Jun 06, 04 15:42 **eg19−pipe.c** Page

```c
/*
 * This program runs ls | wc.
 *
 * Mmm.. what happen if I remove the close()
 * statement?
 */
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>

int main()
{
    int fd[2];
    pid_t pid;

    pipe(fd);
    pid = fork();
    if (pid == 0) {
        // child
        close(fd[1]);
        dup2(fd[0], STDIN_FILENO);
        execlp("wc", "wc", 0);
        _exit(0);
    } else {
        // parent
        close(fd[0]);
        dup2(fd[1], STDOUT_FILENO);
        execlp("ls", "ls", 0);
        wait(NULL);
    }
    return 0;
}
```

```c
/*
 * popen and pclose are two high-level implement
 * for pipes.  Remember to close your pipe.
 */
#include <stdio.h>

int main()
{
    char buf[1024];
    FILE *fin = popen("cal 2004", "r");
    FILE *fout = popen("less", "w");
    while (fgets(buf, 1023, fin) != NULL)
        fprintf(fout, "%s", buf);

    pclose(fin);
    pclose(fout);

    return 0;
}
```



```c
/*
 * Introducing signals, signal(), psignal() and signal
 * handlers.
 */

#include <signal.h>
void print_signal(int sig)
{
    psignal(sig, "caught");
}

int main(int argc, char *argv[])
{
    int i;
    for (i = 0; i < SIGRTMAX;i++)
        signal(i, print_signal);

    while (1);
}
```

```
#!/bin/bash
#
# You can install signal handler in shell script as
# well, using bash's builtin command "trap".
#
# trap is commonly used to clean-up temporary files.
#

trap "do_signal" SIGTSTP SIGQUIT SIGINT

do_signal()
{
    echo "ignoring your command"
}

while :; do
    :
done
```