

testlowlevelout.c (Page 1 of 1)

```
1: /*-----*/
2: /* testlowlevelout.c */
3: /* Author: Bob Dondero */
4: /* The creat, write, and close system calls. */
5: /*-----*/
6:
7: #include <stdio.h>
8: #include <stdlib.h>
9: #include <string.h>
10: #include <unistd.h>
11: #include <fcntl.h>
12:
13: /*-----*/
14:
15: /* Demonstrate the creat(), write(), and close() system-level functions.
16:    As usual, argc is the command-line argument count, and argv contains
17:    the command-line arguments. Return 0. */
18:
19: int main(int argc, char *argv[])
20: {
21:     int iFd;
22:     int iRet;
23:     ssize_t lByteCount;
24:     char *pcBuffer = "somedata\n";
25:
26:     /* The permissions of the newly-created file. */
27:     enum {PERMISSIONS = 0600};
28:
29:     /* Write "somedata\n" to a file named tempfile. */
30:
31:     iFd = creat("tempfile", PERMISSIONS);
32:     if (iFd == -1) {perror(argv[0]); exit(EXIT_FAILURE); }
33:
34:     lByteCount = write(iFd, pcBuffer, strlen(pcBuffer));
35:     if (lByteCount == -1) {perror(argv[0]); exit(EXIT_FAILURE); }
36:
37:     iRet = close(iFd);
38:     if (iRet == -1) {perror(argv[0]); exit(EXIT_FAILURE); }
39:
40:     return 0;
41: }
42:
43: /*-----*/
44:
45: /* Sample execution:
46:
47: $ gcc217 testlowlevelout.c -o testlowlevelout
48:
49: $ ./testlowlevelout
50:
51: $ cat tempfile
52: somedata
53:
54: */
```

testlowlevelin.c (Page 1 of 1)

```

1: /*-----*/
2: /* testlowlevelin.c */
3: /* Author: Bob Dondero */
4: /*-----*/
5:
6: #include <stdio.h>
7: #include <stdlib.h>
8: #include <unistd.h>
9: #include <fcntl.h>
10:
11: /*-----*/
12:
13: /* Demonstrate the open(), read(), and close() system-level functions.
14:    As usual, argc is the command-line argument count, and argv contains
15:    the command-line arguments. Return 0. */
16:
17: int main(int argc, char *argv[])
18: {
19:     enum {BYTE_COUNT = 9};
20:
21:     int iFd;
22:     int iRet;
23:     char acBuffer[BYTE_COUNT] = {'\0'};
24:     int i;
25:     ssize_t lByteCount;
26:
27:     /* Read "somedata\n" from a file named tempfile. */
28:
29:     iFd = open("tempfile", O_RDONLY);
30:     if (iFd == -1) {perror(argv[0]); exit(EXIT_FAILURE); }
31:
32:     lByteCount = read(iFd, acBuffer, BYTE_COUNT);
33:     if (lByteCount == -1) {perror(argv[0]); exit(EXIT_FAILURE);}
34:
35:     iRet = close(iFd);
36:     if (iRet == -1) {perror(argv[0]); exit(EXIT_FAILURE); }
37:
38:     /* Write the data to verify that the previous worked. */
39:
40:     for (i = 0; i < BYTE_COUNT; i++)
41:         putchar(acBuffer[i]);
42:
43:     return 0;
44: }
45:
46: /*-----*/
47:
48: /* Sample execution:
49:
50: $ gcc217 testlowlevelin.c -o testlowlevelin
51:
52: $ ./testlowlevelin
53: somedata
54:
55: */

```

testdupout.c (Page 1 of 1)

```
1: /*-----*/
2: /* testdupout.c */
3: /* Author: Bob Dondero */
4: /*-----*/
5:
6: #include <stdio.h>
7: #include <stdlib.h>
8: #include <unistd.h>
9: #include <fcntl.h>
10:
11: /*-----*/
12:
13: /* Demonstrate the creat(), close(), and dup() system-level functions.
14:    As usual, argc is the command-line argument count, and argv contains
15:    the command-line arguments. Return 0. */
16:
17: int main(int argc, char *argv[])
18: {
19:     int iFd;
20:     int iRet;
21:
22:     /* The permissions of the newly-created file. */
23:     enum {PERMISSIONS = 0600};
24:
25:     iFd = creat("tempfile", PERMISSIONS);
26:     if (iFd == -1) {perror(argv[0]); exit(EXIT_FAILURE); }
27:
28:     iRet = close(1);
29:     if (iRet == -1) {perror(argv[0]); exit(EXIT_FAILURE); }
30:
31:     iRet = dup(iFd);
32:     if (iRet == -1) {perror(argv[0]); exit(EXIT_FAILURE); }
33:
34:     iRet = close(iFd);
35:     if (iRet == -1) {perror(argv[0]); exit(EXIT_FAILURE); }
36:
37:     fputs("somedata\n", stdout);
38:
39:     return 0;
40: }
41:
42: /*-----*/
43:
44: /* Sample execution:
45:
46: $ gcc217 testdupout.c -o testdupout
47:
48: $ ./testdupout
49:
50: $ cat tempfile
51: somedata
52:
53: */
```

testdupin.c (Page 1 of 1)

```

1: /*-----*/
2: /* testdupin.c */
3: /* Author: Bob Dondero */
4: /*-----*/
5:
6: #include <stdio.h>
7: #include <stdlib.h>
8: #include <unistd.h>
9: #include <fcntl.h>
10:
11: /*-----*/
12:
13: /* Demonstrate the open(), close, and dup() system-level functions.
14:    As usual, argc is the command-line argument count, and argv contains
15:    the command-line arguments. Return 0. */
16:
17: int main(int argc, char *argv[])
18: {
19:     enum {BUFFER_LENGTH = 100};
20:     int iFd;
21:     int iRet;
22:     char *pcRet;
23:     char acBuffer[BUFFER_LENGTH];
24:
25:     iFd = open("tempfile", O_RDONLY);
26:     if (iFd == -1) {perror(argv[0]); exit(EXIT_FAILURE); }
27:
28:     iRet = close(0);
29:     if (iRet == -1) {perror(argv[0]); exit(EXIT_FAILURE); }
30:
31:     iRet = dup(iFd);
32:     if (iRet == -1) {perror(argv[0]); exit(EXIT_FAILURE); }
33:
34:     iRet = close(iFd);
35:     if (iRet == -1) {perror(argv[0]); exit(EXIT_FAILURE); }
36:
37:     pcRet = fgets(acBuffer, BUFFER_LENGTH, stdin);
38:     if (pcRet == NULL) {perror(argv[0]); exit(EXIT_FAILURE); }
39:
40:     /* Write the data to verify that the call of fgets worked. */
41:     fputs(acBuffer, stdout);
42:
43:     return 0;
44: }
45:
46: /*-----*/
47:
48: /* Sample execution:
49:
50: $ gcc217 testdupin.c -o testdupin
51:
52: $ ./testdupin
53: somedata
54:
55: */

```

testdupforkexec.c (Page 1 of 2)

```

1: /*-----*/
2: /* testdupforkexec.c */
3: /* Author: Bob Dondero */
4: /*-----*/
5:
6: #include <stdio.h>
7: #include <stdlib.h>
8: #include <unistd.h>
9: #include <fcntl.h>
10: #include <sys/wait.h>
11:
12: /*-----*/
13:
14: /* Demonstrate the fork(), creat(), close(), and dup() system-level
15:    functions. As usual, argc is the command-line argument count, and
16:    argv contains the command-line arguments. Return 0. */
17:
18: int main(int argc, char *argv[])
19: {
20:     /* The Unix permissions that a newly created file should have. */
21:     enum {PERMISSIONS = 0600};
22:
23:     pid_t iPid;
24:     int iRet;
25:
26:     printf("%d parent\n", (int)getpid());
27:
28:     iRet = fflush(stdin);
29:     if (iRet == EOF) {perror(argv[0]); exit(EXIT_FAILURE); }
30:     iRet = fflush(stdout);
31:     if (iRet == EOF) {perror(argv[0]); exit(EXIT_FAILURE); }
32:
33:     iPid = fork();
34:     if (iPid == -1) {perror(argv[0]); exit(EXIT_FAILURE); }
35:
36:     if (iPid == 0)
37:     {
38:         char *apcArgv[2];
39:         int iFd;
40:         int iRet;
41:
42:         iFd = creat("tempfile", PERMISSIONS);
43:         if (iFd == -1) {perror(argv[0]); exit(EXIT_FAILURE); }
44:
45:         iRet = close(1);
46:         if (iRet == -1) {perror(argv[0]); exit(EXIT_FAILURE); }
47:
48:         iRet = dup(iFd);
49:         if (iRet == -1) {perror(argv[0]); exit(EXIT_FAILURE); }
50:
51:         iRet = close(iFd);
52:         if (iRet == -1) {perror(argv[0]); exit(EXIT_FAILURE); }
53:
54:         apcArgv[0] = "date";
55:         apcArgv[1] = NULL;
56:         execvp("date", apcArgv);
57:         perror(argv[0]);
58:         exit(EXIT_FAILURE);
59:     }
60:
61:     iPid = wait(NULL);
62:     if (iPid == -1) {perror(argv[0]); exit(EXIT_FAILURE); }
63:

```

testdupforkexec.c (Page 2 of 2)

```
64:    /* This code is executed by only the parent process. */
65:    printf("%d parent\n", (int)getpid());
66:
67:    return 0;
68: }
69:
70: /*-----*/
71:
72: /* Sample execution:
73:
74: $ gcc217 testdupforkexec.c -o testdupforkexec
75:
76: $ ./testdupforkexec
77: 1140 parent
78: 1140 parent
79:
80: $ cat tempfile
81: Sat Apr 27 17:47:26 EDT 2019
82:
83: */
```

testsignal.c (Page 1 of 2)

```

1: /*-----*/
2: /* testsignal.c */
3: /* Author: Bob Dondero */
4: /*-----*/
5:
6: #define _GNU_SOURCE
7:
8: #include <stdio.h>
9: #include <stdlib.h>
10: #include <string.h>
11: #include <signal.h>
12: #include <unistd.h>
13:
14: /*-----*/
15:
16: /* This function is intended to be a signal handler. Write a message
17:    to stdout that contains iSignal, the number of the signal that
18:    caused the function to be called. */
19:
20: static void myHandler(int iSignal)
21: {
22:     /* Really shouldn't call printf() here. See Bryant & O'Hallaron
23:        page 766. */
24:     printf("In myHandler with parameter %d\n", iSignal);
25: }
26:
27: /*-----*/
28:
29: /* Demonstrate the signal() function and the sigprocmask() system-level
30:    function. As usual, argc is the command-line argument count, and
31:    argv contains the command-line arguments. The function never
32:    returns. */
33:
34: int main(int argc, char *argv[])
35: {
36:     const char *pcPgmName;
37:     void (*pfRet)(int);
38:
39:     pcPgmName = argv[0];
40:
41:     /* Install myHandler as the handler for SIGINT signals. */
42:     pfRet = signal(SIGINT, myHandler);
43:     if (pfRet == SIG_ERR) {perror(pcPgmName); exit(EXIT_FAILURE); }
44:
45:     printf("Entering an infinite loop\n");
46:     for (;;)
47:         sleep(1);
48:
49:     /* Will not reach this point. */
50: }
51:
52: /*-----*/
53:
54: /* Sample execution:
55:
56: $ gcc217 testsignal.c -o testsignal
57:
58: $ ./testsignal
59: Entering an infinite loop
60: ^CIn myHandler with parameter 2
61: ^CIn myHandler with parameter 2
62: ^CIn myHandler with parameter 2
63: ^CIn myHandler with parameter 2

```

testsignal.c (Page 2 of 2)

```
64: ^CIn myHandler with parameter 2
65: ^\Quit
66:
67: */
68:
69: /* Note: Can use kill command or Ctrl-\ to stop process. */
```


testsignalignore.c (Page 1 of 1)

```

1: /*-----*/
2: /* testsignalignore.c */
3: /* Author: Bob Dondero */
4: /*-----*/
5:
6: #define _GNU_SOURCE
7:
8: #include <stdio.h>
9: #include <stdlib.h>
10: #include <signal.h>
11: #include <unistd.h>
12:
13: /*-----*/
14:
15: /* Demonstrate ignoring signals and restoring default signal behavior.
16:    As usual, argc is the command-line argument count, and argv contains
17:    the command-line arguments. The function never returns. */
18:
19: int main(int argc, char *argv[])
20: {
21:     enum {SLEEP_SECONDS = 3};
22:
23:     void (*pfRet)(int);
24:
25:     for (;;)
26:     {
27:         printf("\nFor the next %d seconds ", SLEEP_SECONDS);
28:         printf("SIGINT signals are ignored.\n");
29:         /* Ignore SIGINT signals. */
30:         pfRet = signal(SIGINT, SIG_IGN);
31:         if (pfRet == SIG_ERR) {perror(argv[0]); exit(EXIT_FAILURE); }
32:         sleep(SLEEP_SECONDS);
33:
34:         printf("\nFor the next %d seconds ", SLEEP_SECONDS);
35:         printf("SIGINT signals are not ignored.\n");
36:         /* Restore the default behavior for SIGINT signals. */
37:         pfRet = signal(SIGINT, SIG_DFL);
38:         if (pfRet == SIG_ERR) {perror(argv[0]); exit(EXIT_FAILURE); }
39:         sleep(SLEEP_SECONDS);
40:     }
41:
42:     /* Will not reach this point. */
43: }
44:
45: /*-----*/
46:
47: /* Sample execution:
48:
49: $ gcc217 testsignalignore.c -o testsignalignore
50:
51: $ ./testsignalignore
52:
53: For the next 3 sec SIGINT signals are ignored.
54: ^C^C^C
55: For the next 3 sec SIGINT signals are not ignored.
56:
57: For the next 3 sec SIGINT signals are ignored.
58: ^C^C^C^C
59: For the next 3 sec SIGINT signals are not ignored.
60: ^C
61:
62: */

```

testalarm.c (Page 1 of 2)

```

1: /*-----*/
2: /* testalarm.c */
3: /* Author: Bob Dondero */
4: /*-----*/
5:
6: #define _GNU_SOURCE
7:
8: #include <stdio.h>
9: #include <stdlib.h>
10: #include <string.h>
11: #include <signal.h>
12: #include <unistd.h>
13:
14: /*-----*/
15:
16: /* This function is intended to be a handler for signals of type
17:    SIGALRM. Write a message to stdout that contains iSignal, the
18:    number of the signal that caused the function to be called. Then
19:    reset the alarm. */
20:
21: static void myHandler(int iSignal)
22: {
23:     /* Really shouldn't call printf() here. See Bryant & O'Hallaron
24:        page 766. */
25:     printf("In myHandler with parameter %d.\n", iSignal);
26:
27:     /* Reset the alarm. */
28:     alarm(2);
29: }
30:
31: /*-----*/
32:
33: /* Demonstrate the alarm() function. As usual, argc is the
34:    command-line argument count, and argv contains the command-line
35:    arguments. The function never returns. */
36:
37: int main(int argc, char *argv[])
38: {
39:     const char *pcPgmName;
40:     void (*pfRet)(int);
41:
42:     pcPgmName = argv[0];
43:
44:     /* Install myHandler as the handler for SIGALRM signals. */
45:     pfRet = signal(SIGALRM, myHandler);
46:     if (pfRet == SIG_ERR) {perror(pcPgmName); exit(EXIT_FAILURE); }
47:
48:     /* Set a 2 second alarm. After 2 seconds of real time, send a
49:        SIGALRM signal to this process. */
50:     alarm(2);
51:
52:     /* Enter an infinite loop. */
53:     printf("Entering an infinite loop\n");
54:     for (;;)
55:         sleep(1);
56:
57:     /* Will not reach this point. */
58: }
59:
60: /*-----*/
61:
62: /* Sample execution:
63:

```

testalarm.c (Page 2 of 2)

```
64: $ gcc217 testalarm.c -o testalarm
65:
66: $ ./testalarm
67: Entering an infinite loop
68: In myHandler with parameter 14.
69: In myHandler with parameter 14.
70: In myHandler with parameter 14.
71: In myHandler with parameter 14.
72: ^C
73:
74: */
```

testalarmtimeout.c (Page 1 of 2)

```

1: /*-----*/
2: /* testalarmtimeout.c */
3: /* Author: Bob Dondero */
4: /*-----*/
5:
6: #define _GNU_SOURCE
7:
8: #include <stdio.h>
9: #include <stdlib.h>
10: #include <string.h>
11: #include <signal.h>
12: #include <unistd.h>
13:
14: /*-----*/
15:
16: /* This function is intended to be a handler for signals of type
17:    SIGALRM. Write a timeout message to stdout, and exit. iSignal is
18:    number of the signal that caused this handler to execute. */
19:
20: static void myHandler(int iSignal)
21: {
22:     /* Really shouldn't call printf() or exit() here. See Bryant &
23:        O'Hallaron page 766. */
24:     printf("\nSorry. You took too long.\n");
25:     exit(EXIT_FAILURE);
26: }
27:
28: /*-----*/
29:
30: /* Demonstrate using an alarm to cause a timeout. As usual, argc is
31:    the command-line argument count, and argv contains the command-line
32:    arguments. Return 0. */
33:
34: int main(int argc, char *argv[])
35: {
36:     enum {ALARM_DURATION_SECONDS = 5};
37:
38:     const char *pcPgmName;
39:     int i;
40:     void (*pfRet)(int);
41:
42:     pcPgmName = argv[0];
43:
44:     /* Install myHandler as the handler for SIGALRM signals. */
45:     pfRet = signal(SIGALRM, myHandler);
46:     if (pfRet == SIG_ERR) {perror(pcPgmName); exit(EXIT_FAILURE); }
47:
48:     printf("Enter a number: ");
49:
50:     alarm(ALARM_DURATION_SECONDS);
51:     scanf("%d", &i);
52:     alarm(0);
53:
54:     printf("You entered the number %d.\n", i);
55:     return 0;
56: }
57:
58: /*-----*/
59:
60: /* Sample execution:
61:
62: $ gcc217 testalarmtimeout.c -o testalarmtimeout
63:

```

testalarmtimeout.c (Page 2 of 2)

```
64: $ ./testalarmtimeout
65: Enter a number: 123
66: You entered the number 123.
67:
68: $ ./testalarmtimeout
69: Enter a number:
70: Sorry. You took too long.
71:
72: */
```