testlowlevelout.c (Page 1 of 1)

```c
/*--------------------------------------------------------------------*/
/* testlowlevelout.c                                                  */
/* Author: Bob Dondero                                                */
/* The creat, write, and close system calls.                          */
/*--------------------------------------------------------------------*/

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

/*--------------------------------------------------------------------*/

/* Demonstrate the creat(), write(), and close() system-level functions.
   As usual, argc is the command-line argument count, and argv contains
   the command-line arguments. Return 0. */

int main(int argc, char *argv[])
{
   int iFd;
   int iRet;
   ssize_t lByteCount;
   char *pcBuffer = "somedata\n";

   /* The permissions of the newly-created file. */
   enum {PERMISSIONS = 0600};

   /* Write "somedata\n" to a file named tempfile. */

   iFd = creat("tempfile", PERMISSIONS);
   if (iFd == -1) {perror(argv[0]); exit(EXIT_FAILURE); }

   lByteCount = write(iFd, pcBuffer, strlen(pcBuffer));
   if (lByteCount == -1) {perror(argv[0]); exit(EXIT_FAILURE); }

   iRet = close(iFd);
   if (iRet == -1) {perror(argv[0]); exit(EXIT_FAILURE); }

   return 0;
}

/*--------------------------------------------------------------------*/

/* Sample execution:

$ gcc217 testlowlevelout.c -o testlowlevelout

$ ./testlowlevelout

$ cat tempfile
somedata

*/
```

testlowlevelin.c (Page 1 of 1)

```c
/*--------------------------------------------------------------------*/
/* testlowlevelin.c                                                   */
/* Author: Bob Dondero                                                */
/*--------------------------------------------------------------------*/

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

/*--------------------------------------------------------------------*/

/* Demonstrate the open(), read(), and close() system-level functions.
   As usual, argc is the command-line argument count, and argv contains
   the command-line arguments. Return 0. */

int main(int argc, char *argv[])
{
   enum {BYTE_COUNT = 9};

   int iFd;
   int iRet;
   char acBuffer[BYTE_COUNT] = {'\0'};
   int i;
   ssize_t lByteCount;

   /* Read "somedata\n" from a file named tempfile. */

   iFd = open("tempfile", O_RDONLY);
   if (iFd == -1) {perror(argv[0]); exit(EXIT_FAILURE); }

   lByteCount = read(iFd, acBuffer, BYTE_COUNT);
   if (lByteCount == -1) {perror(argv[0]); exit(EXIT_FAILURE);}

   iRet = close(iFd);
   if (iRet == -1) {perror(argv[0]); exit(EXIT_FAILURE); }

   /* Write the data to verify that the previous worked. */

   for (i = 0; i < BYTE_COUNT; i++)
      putchar(acBuffer[i]);

   return 0;
}

/*--------------------------------------------------------------------*/

/* Sample execution:

$ gcc217 testlowlevelin.c -o testlowlevelin

$ ./testlowlevelin
somedata

*/
```

testdupout.c (Page 1 of 1)

```c
/*--------------------------------------------------------------------*/
/* testdupout.c                                                       */
/* Author: Bob Dondero                                                */
/*--------------------------------------------------------------------*/

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

/*--------------------------------------------------------------------*/

/* Demonstrate the creat(), close(), and dup() system-level functions.
   As usual, argc is the command-line argument count, and argv contains
   the command-line arguments. Return 0. */

int main(int argc, char *argv[])
{
   int iFd;
   int iRet;

   /* The permissions of the newly-created file. */
   enum {PERMISSIONS = 0600};

   iFd = creat("tempfile", PERMISSIONS);
   if (iFd == -1) {perror(argv[0]); exit(EXIT_FAILURE); }

   iRet = close(1);
   if (iRet == -1) {perror(argv[0]); exit(EXIT_FAILURE); }

   iRet = dup(iFd);
   if (iRet == -1) {perror(argv[0]); exit(EXIT_FAILURE); }

   iRet = close(iFd);
   if (iRet == -1) {perror(argv[0]); exit(EXIT_FAILURE); }

   fputs("somedata\n", stdout);

   return 0;
}

/*--------------------------------------------------------------------*/

/* Sample execution:

$ gcc217 testdupout.c -o testdupout

$ ./testdupout

$ cat tempfile
somedata

*/
```

testdupin.c (Page 1 of 1)

```c
/*--------------------------------------------------------------------*/
/* testdupin.c                                                        */
/* Author: Bob Dondero                                                */
/*--------------------------------------------------------------------*/

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

/*--------------------------------------------------------------------*/

/* Demonstrate the open(), close, and dup() system-level functions.
   As usual, argc is the command-line argument count, and argv contains
   the command-line arguments.  Return 0. */

int main(int argc, char *argv[])
{
   enum {BUFFER_LENGTH = 100};
   int iFd;
   int iRet;
   char *pcRet;
   char acBuffer[BUFFER_LENGTH];

   iFd = open("tempfile", O_RDONLY);
   if (iFd == -1) {perror(argv[0]); exit(EXIT_FAILURE); }

   iRet = close(0);
   if (iRet == -1) {perror(argv[0]); exit(EXIT_FAILURE); }

   iRet = dup(iFd);
   if (iRet == -1) {perror(argv[0]); exit(EXIT_FAILURE); }

   iRet = close(iFd);
   if (iRet == -1) {perror(argv[0]); exit(EXIT_FAILURE); }

   pcRet = fgets(acBuffer, BUFFER_LENGTH, stdin);
   if (pcRet == NULL) {perror(argv[0]); exit(EXIT_FAILURE); }

   /* Write the data to verify that the call of fgets worked. */
   fputs(acBuffer, stdout);

   return 0;
}

/*--------------------------------------------------------------------*/

/* Sample execution:

$ gcc217 testdupin.c -o testdupin

$ ./testdupin
somedata

*/
```

testdupforkexec.c (Page 1 of 2)

```c
/*--------------------------------------------------------------------*/
/* testdupforkexec.c                                                  */
/* Author: Bob Dondero                                                */
/*--------------------------------------------------------------------*/

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/wait.h>

/*--------------------------------------------------------------------*/

/* Demonstrate the fork(), creat(), close(), and dup() system-level
   functions. As usual, argc is the command-line argument count, and
   argv contains the command-line arguments. Return 0. */

int main(int argc, char *argv[])
{
   /* The Unix permissions that a newly created file should have. */
   enum {PERMISSIONS = 0600};

   pid_t iPid;
   int iRet;

   printf("%d parent\n", (int)getpid());

   iRet = fflush(stdin);
   if (iRet == EOF) {perror(argv[0]); exit(EXIT_FAILURE); }
   iRet = fflush(stdout);
   if (iRet == EOF) {perror(argv[0]); exit(EXIT_FAILURE); }

   iPid = fork();
   if (iPid == -1) {perror(argv[0]); exit(EXIT_FAILURE); }

   if (iPid == 0)
   {
      char *apcArgv[2];
      int iFd;
      int iRet;

      iFd = creat("tempfile", PERMISSIONS);
      if (iFd == -1) {perror(argv[0]); exit(EXIT_FAILURE); }

      iRet = close(1);
      if (iRet == -1) {perror(argv[0]); exit(EXIT_FAILURE); }

      iRet = dup(iFd);
      if (iRet == -1) {perror(argv[0]); exit(EXIT_FAILURE); }

      iRet = close(iFd);
      if (iRet == -1) {perror(argv[0]); exit(EXIT_FAILURE); }

      apcArgv[0] = "date";
      apcArgv[1] = NULL;
      execvp("date", apcArgv);
      perror(argv[0]);
      exit(EXIT_FAILURE);
   }

   iPid = wait(NULL);
   if (iPid == -1) {perror(argv[0]); exit(EXIT_FAILURE); }

```

testdupforkexec.c (Page 2 of 2)

```
   /* This code is executed by only the parent process. */
   printf("%d parent\n", (int)getpid());

   return 0;
}

/*----------------------------------------------------------------------*/

/* Sample execution:

$ gcc217 testdupforkexec.c -o testdupforkexec

$ ./testdupforkexec
1140 parent
1140 parent

$ cat tempfile
Sat Apr 27 17:47:26 EDT 2019

*/
```

testsignal.c (Page 1 of 2)

```
/*--------------------------------------------------------------------*/
/* testsignal.c                                                       */
/* Author: Bob Dondero                                                */
/*--------------------------------------------------------------------*/

#define _GNU_SOURCE

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <signal.h>
#include <unistd.h>

/*--------------------------------------------------------------------*/

/* This function is intended to be a signal handler.  Write a message
   to stdout that contains iSignal, the number of the signal that
   caused the function to be called. */

static void myHandler(int iSignal)
{
   /* Really shouldn't call printf() here. See Bryant & O'Hallaron
      page 766. */
   printf("In myHandler with parameter %d\n", iSignal);
}

/*--------------------------------------------------------------------*/

/* Demonstrate the signal() function and the sigprocmask() system-levl
   function. As usual, argc is the command-line argument count, and
   argv contains the command-line arguments. The function never
   returns. */

int main(int argc, char *argv[])
{
   const char *pcPgmName;
   void (*pfRet)(int);

   pcPgmName = argv[0];

   /* Install myHandler as the handler for SIGINT signals. */
   pfRet = signal(SIGINT, myHandler);
   if (pfRet == SIG_ERR) {perror(pcPgmName); exit(EXIT_FAILURE); }

   printf("Entering an infinite loop\n");
   for (;;)
      sleep(1);

   /* Will not reach this point. */
}

/*--------------------------------------------------------------------*/

/* Sample execution:

$ gcc217 testsignal.c -o testsignal

$ ./testsignal
Entering an infinite loop
^CIn myHandler with parameter 2
^CIn myHandler with parameter 2
^CIn myHandler with parameter 2
^CIn myHandler with parameter 2
```

testsignal.c (Page 2 of 2)

```
^CIn myHandler with parameter 2
^\Quit

*/

/* Note:  Can use kill command or Ctrl-\ to stop process. */
```

testsignalignore.c (Page 1 of 1)

```c
/*--------------------------------------------------------------------*/
/* testsignalignore.c                                                 */
/* Author: Bob Dondero                                                */
/*--------------------------------------------------------------------*/

#define _GNU_SOURCE

#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <unistd.h>

/*--------------------------------------------------------------------*/

/* Demonstrate ignoring signals and restoring default signal behavior.
   As usual, argc is the command-line argument count, and argv contains
   the command-line arguments. The function never returns. */

int main(int argc, char *argv[])
{
   enum {SLEEP_SECONDS = 3};

   void (*pfRet)(int);

   for (;;)
   {
      printf("\nFor the next %d seconds ", SLEEP_SECONDS);
      printf("SIGINT signals are ignored.\n");
      /* Ignore SIGINT signals. */
      pfRet = signal(SIGINT, SIG_IGN);
      if (pfRet == SIG_ERR) {perror(argv[0]); exit(EXIT_FAILURE); }
      sleep(SLEEP_SECONDS);

      printf("\nFor the next %d seconds ", SLEEP_SECONDS);
      printf("SIGINT signals are not ignored.\n");
      /* Restore the default behavior for SIGINT signals. */
      pfRet = signal(SIGINT, SIG_DFL);
      if (pfRet == SIG_ERR) {perror(argv[0]); exit(EXIT_FAILURE); }
      sleep(SLEEP_SECONDS);
   }

   /* Will not reach this point. */
}

/*--------------------------------------------------------------------*/

/* Sample execution:

$ gcc217 testsignalignore.c -o testsignalignore

$ ./testsignalignore

For the next 3 sec SIGINT signals are ignored.
^C^C^C
For the next 3 sec SIGINT signals are not ignored.

For the next 3 sec SIGINT signals are ignored.
^C^C^C^C
For the next 3 sec SIGINT signals are not ignored.
^C

*/
```

testalarm.c (Page 1 of 2)

```c
/*--------------------------------------------------------------------*/
/* testalarm.c                                                        */
/* Author: Bob Dondero                                                */
/*--------------------------------------------------------------------*/

#define _GNU_SOURCE

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <signal.h>
#include <unistd.h>

/*--------------------------------------------------------------------*/

/* This function is intended to be a handler for signals of type
   SIGALRM.  Write a message to stdout that contains iSignal, the
   number of the signal that caused the function to be called.  Then
   reset the alarm. */

static void myHandler(int iSignal)
{
   /* Really shouldn't call printf() here. See Bryant & O'Hallaron
      page 766. */
   printf("In myHandler with parameter %d.\n", iSignal);

   /* Reset the alarm. */
   alarm(2);
}

/*--------------------------------------------------------------------*/

/* Demonstrate the alarm() function.  As usual, argc is the
   command-line argument count, and argv contains the command-line
   arguments.  The function never returns. */

int main(int argc, char *argv[])
{
   const char *pcPgmName;
   void (*pfRet)(int);

   pcPgmName = argv[0];

   /* Install myHandler as the handler for SIGALRM signals. */
   pfRet = signal(SIGALRM, myHandler);
   if (pfRet == SIG_ERR) {perror(pcPgmName); exit(EXIT_FAILURE); }

   /* Set a 2 second alarm.  After 2 seconds of real time, send a
      SIGALRM signal to this process. */
   alarm(2);

   /* Enter an infinite loop. */
   printf("Entering an infinite loop\n");
   for (;;)
      sleep(1);

   /* Will not reach this point. */
}

/*--------------------------------------------------------------------*/

/* Sample execution:

```

testalarm.c (Page 2 of 2)

```
$ gcc217 testalarm.c -o testalarm

$ ./testalarm
Entering an infinite loop
In myHandler with parameter 14.
In myHandler with parameter 14.
In myHandler with parameter 14.
In myHandler with parameter 14.
^C

*/
```

testalarmtimeout.c (Page 1 of 2)

```c
/*--------------------------------------------------------------------*/
/* testalarmtimeout.c                                                 */
/* Author: Bob Dondero                                                */
/*--------------------------------------------------------------------*/

#define _GNU_SOURCE

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <signal.h>
#include <unistd.h>

/*--------------------------------------------------------------------*/

/* This function is intended to be a handler for signals of type
   SIGALRM. Write a timeout message to stdout, and exit. iSignal is
   number of the signal that caused this handler to execute. */

static void myHandler(int iSignal)
{
   /* Really shouldn't call printf() or exit() here. See Bryant &
      O'Hallaron page 766. */
   printf("\nSorry. You took too long.\n");
   exit(EXIT_FAILURE);
}

/*--------------------------------------------------------------------*/

/* Demonstrate using an alarm to cause a timeout. As usual, argc is
   the command-line argument count, and argv contains the command-line
   arguments. Return 0. */

int main(int argc, char *argv[])
{
   enum {ALARM_DURATION_SECONDS = 5};

   const char *pcPgmName;
   int i;
   void (*pfRet)(int);

   pcPgmName = argv[0];

   /* Install myHandler as the handler for SIGALRM signals. */
   pfRet = signal(SIGALRM, myHandler);
   if (pfRet == SIG_ERR) {perror(pcPgmName); exit(EXIT_FAILURE); }

   printf("Enter a number: ");

   alarm(ALARM_DURATION_SECONDS);
   scanf("%d", &i);
   alarm(0);

   printf("You entered the number %d.\n", i);
   return 0;
}

/*--------------------------------------------------------------------*/

/* Sample execution:

$ gcc217 testalarmtimeout.c -o testalarmtimeout

$ ./testalarmtimeout
Enter a number: 123
You entered the number 123.

$ ./testalarmtimeout
Enter a number:
Sorry. You took too long.

*/
```