

hello.c (Page 1 of 1)

```
1: /*-----*/
2: /* hello.c */
3: /* Author: Bob Dondero */
4: /*-----*/
5:
6: #include <stdio.h>
7: #include <unistd.h>
8: #include <sys/types.h>
9:
10: /*-----*/
11:
12: /* Demonstrate the system-level getpid() function. Return 0. */
13:
14: int main(void)
15: {
16:     pid_t iPid;
17:     iPid = getpid();
18:     printf("%d hello\n", (int)iPid);
19:     return 0;
20: }
21:
22: /*-----*/
23:
24: /* Sample executions:
25:
26: $ gcc217 hello.c -o hello
27:
28: $ ./hello
29: 22386 hello
30:
31: $ ./hello
32: 22387 hello
33:
34: */
```

testexec.c (Page 1 of 1)

```
1: /*-----*/
2: /* textexec.c */
3: /* Author: Bob Dondero */
4: /*-----*/
5:
6: #include <stdio.h>
7: #include <stdlib.h>
8: #include <unistd.h>
9:
10: /*-----*/
11:
12: /* Demonstrate the system-level execvp() function and the
13:  standard C perror() function. Return nothing if successful, and
14:  EXIT_FAILURE upon failure. As usual, argc is the command-line
15:  argument count, and argv contains the command-line arguments. */
16:
17: int main(int argc, char *argv[])
18: {
19:     char *apcArgv[2];
20:
21:     printf("%d testexec\n", (int)getpid());
22:
23:     apcArgv[0] = "./hello";
24:     apcArgv[1] = NULL;
25:
26:     execvp("./hello", apcArgv);
27:
28:     perror(argv[0]);
29:     /* Alternative:
30:        fprintf(stderr, "%s: %s\n", argv[0], strerror(errno)); */
31:
32:     exit(EXIT_FAILURE);
33: }
34:
35: /*-----*/
36:
37: /* Sample executions:
38:
39: $ gcc217 hello.c -o hello
40:
41: $ gcc217 testexec.c -o testexec
42:
43: $ ./testexec
44: 22440 testexec
45: 22440 hello
46:
47: $ mv hello nothello
48:
49: $ ./testexec
50: 22454 testexec
51: ./testexec: No such file or directory
52:
53: $ mv nothello hello
54:
55: $ ./testexec
56: 22463 testexec
57: 22463 hello
58:
59: */
```

Princeton University
 COS 217: Introduction to Programming Systems
 The exec Linux System-Level Functions

System Call	Searches PATH (p)?	Accepts Vector (v)?	Accepts Environment (e)?
int execl (const char *path, const char *arg0, ..., const char *argn, char * /*NULL*/);	No	No	No
int execv (const char *path, char *const argv[]);	No	Yes	No
int execle (const char *path, const char *arg0, ..., const char *argn, char * /*NULL*/, char *const envp[]);	No	No	Yes
int execve (const char *path, char *const argv[], char *const envp[]);	No	Yes	Yes
int execlp (const char *file, const char *arg0, ..., const char *argn, char * /*NULL*/);	Yes	No	No
int execvp (const char *file, char *const argv[]);	Yes	Yes	No

testfork.c (Page 1 of 1)

```

1: /*-----*/
2: /* testfork.c */
3: /* Author: Bob Dondero */
4: /*-----*/
5:
6: #include <stdio.h>
7: #include <stdlib.h>
8: #include <unistd.h>
9: #include <sys/types.h>
10:
11: /*-----*/
12:
13: /* Demonstrate the system-level fork() function and the
14:    standard C fflush() function. Return 0.
15:    As usual, argc is the command-line argument count, and argv
16:    contains the command-line arguments. */
17:
18: int main(int argc, char *argv[])
19: {
20:     pid_t iPid;
21:     int iRet;
22:
23:     printf("%d parent\n", (int)getpid());
24:
25:     iRet = fflush(stdin);
26:     if (iRet == EOF) {perror(argv[0]); exit(EXIT_FAILURE); }
27:     iRet = fflush(stdout);
28:     if (iRet == EOF) {perror(argv[0]); exit(EXIT_FAILURE); }
29:
30:     iPid = fork();
31:     if (iPid == -1) {perror(argv[0]); exit(EXIT_FAILURE); }
32:
33:     printf("%d parent and child\n", (int)getpid());
34:
35:     return 0;
36: }
37:
38: /*-----*/
39:
40: /* Sample executions:
41:
42: $ gcc217 testfork.c -o testfork
43:
44: $ ./testfork
45: 29285 parent
46: 29285 parent and child
47: 29286 parent and child
48:
49: $ ./testfork
50: 29287 parent
51: 29287 parent and child
52: 29288 parent and child
53:
54: */

```

testforkret.c (Page 1 of 2)

```

1: /*-----*/
2: /* testforkret.c */
3: /* Author: Bob Dondero */
4: /*-----*/
5:
6: #include <stdio.h>
7: #include <stdlib.h>
8: #include <unistd.h>
9: #include <sys/types.h>
10:
11: /*-----*/
12:
13: /* Demonstrate how to use the value returned by the system-level
14:    fork() function. Return 0;
15:    As usual, argc is the command-line argument count, and argv
16:    contains the command-line arguments. */
17:
18: int main(int argc, char *argv[])
19: {
20:     pid_t iPid;
21:     int iRet;
22:
23:     printf("%d parent\n", (int)getpid());
24:
25:     iRet = fflush(stdin);
26:     if (iRet == EOF) {perror(argv[0]); exit(EXIT_FAILURE); }
27:     iRet = fflush(stdout);
28:     if (iRet == EOF) {perror(argv[0]); exit(EXIT_FAILURE); }
29:
30:     iPid = fork();
31:     if (iPid == -1) {perror(argv[0]); exit(EXIT_FAILURE); }
32:
33:     if (iPid == 0)
34:         /* This code is executed by the child process only. */
35:         printf("%d child\n", (int)getpid());
36:     else
37:         /* This code is executed by the parent process only. */
38:         printf("%d parent\n", (int)getpid());
39:
40:     /* This code is executed by both the parent and child processes. */
41:     printf("%d parent and child\n", (int)getpid());
42:
43:     return 0;
44: }
45:
46: /*-----*/
47:
48: /* Sample executions:
49:
50: $ gcc217 testforkret.c -o testforkret
51:
52: $ ./testforkret
53: 29409 parent
54: 29409 parent
55: 29409 parent and child
56: 29410 child
57: 29410 parent and child
58:
59: $ ./testforkret
60: 29413 parent
61: 29413 parent
62: 29413 parent and child
63: 29414 child

```

testforkret.c (Page 2 of 2)

```
64: 29414 parent and child  
65:  
66: */
```

testforkexit.c (Page 1 of 1)

```

1: /*-----*/
2: /* testforkexit.c */
3: /* Author: Bob Dondero */
4: /*-----*/
5:
6: #include <stdio.h>
7: #include <stdlib.h>
8: #include <unistd.h>
9: #include <sys/types.h>
10:
11: /*-----*/
12:
13: /* Demonstrate the common pattern of using the system-level fork()
14:    function and the standard C exit() function. Return 0.
15:    As usual, argc is the command-line argument count, and argv
16:    contains the command-line arguments. */
17:
18: int main(int argc, char *argv[])
19: {
20:     pid_t iPid;
21:     int iRet;
22:
23:     printf("%d parent\n", (int)getpid());
24:
25:     iRet = fflush(stdin);
26:     if (iRet == EOF) {perror(argv[0]); exit(EXIT_FAILURE); }
27:     iRet = fflush(stdout);
28:     if (iRet == EOF) {perror(argv[0]); exit(EXIT_FAILURE); }
29:
30:     iPid = fork();
31:     if (iPid == -1) {perror(argv[0]); exit(EXIT_FAILURE); }
32:
33:     if (iPid == 0)
34:     {
35:         /* This code is executed by the child process only. */
36:         printf("%d child\n", (int)getpid());
37:         exit(0);
38:     }
39:
40:     /* This code is executed by the parent process only. */
41:     printf("%d parent\n", (int)getpid());
42:
43:     return 0;
44: }
45:
46: /*-----*/
47:
48: /* Sample executions:
49:
50: $ gcc217 testforkexit.c -o testforkexit
51:
52: $ ./testforkexit
53: 29595 parent
54: 29595 parent
55: 29596 child
56:
57: $ ./testforkexit
58: 29600 parent
59: 29600 parent
60: 29601 child
61:
62: */

```

testforkloop.c (Page 1 of 2)

```

1: /*-----*/
2: /* testforkloop.c */
3: /* Author: Bob Dondero */
4: /*-----*/
5:
6: #include <stdio.h>
7: #include <stdlib.h>
8: #include <unistd.h>
9: #include <sys/types.h>
10:
11: /*-----*/
12:
13: /* Demonstrate context switching among concurrent processes. Return 0.
14:    As usual, argc is the command-line argument count, and argv
15:    contains the command-line arguments. */
16:
17: int main(int argc, char *argv[])
18: {
19:     pid_t iPid;
20:     int iRet;
21:     int i;
22:
23:     printf("%d parent\n", (int)getpid());
24:
25:     iRet = fflush(stdin);
26:     if (iRet == EOF) {perror(argv[0]); exit(EXIT_FAILURE); }
27:     iRet = fflush(stdout);
28:     if (iRet == EOF) {perror(argv[0]); exit(EXIT_FAILURE); }
29:
30:     iPid = fork();
31:     if (iPid == -1) {perror(argv[0]); exit(EXIT_FAILURE); }
32:
33:     if (iPid == 0)
34:     {
35:         /* This code is executed by the child process only. */
36:         for (i = 0; i < 10 ; i++)
37:             printf("%d child %d\n", (int)getpid(), i);
38:         exit(0);
39:     }
40:
41:     /* This code is executed by the parent process only. */
42:     for (i = 0; i < 10; i++)
43:         printf("%d parent %d\n", (int)getpid(), i);
44:
45:     return 0;
46: }
47:
48: /*-----*/
49:
50: /* Sample execution:
51:
52: $ gcc217 testforkloop.c -o testforkloop
53:
54: $ ./testforkloop
55: 9857 parent
56: 9857 parent 0
57: 9857 parent 1
58: 9857 parent 2
59: 9857 parent 3
60: 9857 parent 4
61: 9857 parent 5
62: 9857 parent 6
63: 9857 parent 7

```


testforkloop.c (Page 2 of 2)

```
64: 9858 child 0
65: 9857 parent 8
66: 9858 child 1
67: 9857 parent 9
68: 9858 child 2
69: 9858 child 3
70: 9858 child 4
71: 9858 child 5
72: 9858 child 6
73: 9858 child 7
74: 9858 child 8
75: 9858 child 9
76:
77: */
```

testforkwait.c (Page 1 of 2)

```

1: /*-----*/
2: /* testforkwait.c */
3: /* Author: Bob Dondero */
4: /*-----*/
5:
6: #include <stdio.h>
7: #include <stdlib.h>
8: #include <unistd.h>
9: #include <sys/types.h>
10: #include <sys/wait.h>
11:
12: /*-----*/
13:
14: /* Demonstrate the system-level fork() and wait() functions.
15:    Return 0.
16:    As usual, argc is the command-line argument count, and argv
17:    contains the command-line arguments. */
18:
19: int main(int argc, char *argv[])
20: {
21:     pid_t iPid;
22:     int iRet;
23:     int i = 0;
24:
25:     printf("%d parent\n", (int)getpid());
26:
27:     iRet = fflush(stdin);
28:     if (iRet == EOF) {perror(argv[0]); exit(EXIT_FAILURE); }
29:     iRet = fflush(stdout);
30:     if (iRet == EOF) {perror(argv[0]); exit(EXIT_FAILURE); }
31:
32:     iPid = fork();
33:     if (iPid == -1) {perror(argv[0]); exit(EXIT_FAILURE); }
34:
35:     if (iPid == 0)
36:     {
37:         /* This code is executed by the child process only. */
38:         for (i = 0; i < 10; i++)
39:             printf("%d child %d\n", (int)getpid(), i);
40:         exit(0);
41:     }
42:
43:     /* This code is executed by the parent process only. */
44:
45:     /* Wait for the child process to terminate. */
46:     iPid = wait(NULL);
47:     if (iPid == -1) {perror(argv[0]); exit(EXIT_FAILURE); }
48:
49:     for (i = 0; i < 10; i++)
50:         printf("%d parent %d\n", (int)getpid(), i);
51:
52:     return 0;
53: }
54:
55: /*-----*/
56:
57: /* Sample execution:
58:
59: $ gcc217 testforkwait.c -o testforkwait
60:
61: $ ./testforkwait
62: 7275 parent
63: 7276 child 0

```

testforkwait.c (Page 2 of 2)

```
64: 7276 child 1
65: 7276 child 2
66: 7276 child 3
67: 7276 child 4
68: 7276 child 5
69: 7276 child 6
70: 7276 child 7
71: 7276 child 8
72: 7276 child 9
73: 7275 parent 0
74: 7275 parent 1
75: 7275 parent 2
76: 7275 parent 3
77: 7275 parent 4
78: 7275 parent 5
79: 7275 parent 6
80: 7275 parent 7
81: 7275 parent 8
82: 7275 parent 9
83:
84: */
```

testforkexecwait.c (Page 1 of 2)

```

1: /*-----*/
2: /* textforkexecwait.c */
3: /* Author: Bob Dondero */
4: /*-----*/
5:
6: #include <stdio.h>
7: #include <stdlib.h>
8: #include <unistd.h>
9: #include <sys/types.h>
10: #include <sys/wait.h>
11:
12: /*-----*/
13:
14: /* Demonstrate the common pattern of using the system-level fork(),
15:    execvp(), and wait() functions. Return 0.
16:    As usual, argc is the command-line argument count, and argv
17:    contains the command-line arguments. */
18:
19: int main(int argc, char *argv[])
20: {
21:     enum {SLEEP_SECONDS = 3};
22:
23:     pid_t iPid;
24:     int iRet;
25:
26:     for (;;)
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:             /* This code is executed by the child process only. */
39:             char *apcArgv[2];
40:
41:             apcArgv[0] = "date";
42:             apcArgv[1] = NULL;
43:
44:             execvp("date", apcArgv);
45:             perror(argv[0]);
46:             exit(EXIT_FAILURE);
47:         }
48:
49:         /* This code is executed by the parent process only. */
50:
51:         /* Wait for the child process to exit. */
52:         iPid = wait(NULL);
53:         if (iPid == -1) {perror(argv[0]); exit(EXIT_FAILURE); }
54:
55:         /* Pause for SLEEP_SECONDS seconds. */
56:         sleep(SLEEP_SECONDS);
57:     }
58:
59:     /* Should not reach this point. */
60: }
61:
62: /*-----*/
63:

```

testforkexecwait.c (Page 2 of 2)

```
64: /* Sample execution:
65:
66: $ gcc217 testforkexecwait.c -o testforkexecwait
67:
68: $ ./testforkexecwait
69: Wed Apr 24 21:26:13 EDT 2019
70: Wed Apr 24 21:26:16 EDT 2019
71: Wed Apr 24 21:26:19 EDT 2019
72: Wed Apr 24 21:26:22 EDT 2019
73: Wed Apr 24 21:26:25 EDT 2019
74: Wed Apr 24 21:26:28 EDT 2019
75: Wed Apr 24 21:26:31 EDT 2019
76: Wed Apr 24 21:26:34 EDT 2019
77: ^C
78:
79: */
```

Princeton University
 COS 217: Introduction to Programming Systems
 Linux Orphan and Zombie Processes

