hello.c (Page 1 of 1)

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

#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>

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

/* Demonstrate the system-level getpid() function. Return 0. */

int main(void)
{
   pid_t iPid;
   iPid = getpid();
   printf("%d hello\n", (int)iPid);
   return 0;
}

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

/* Sample executions:

$ gcc217 hello.c -o hello

$ ./hello
22386 hello

$ ./hello
22387 hello

*/
```

testexec.c (Page 1 of 1)

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

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

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

/* Demonstrate the system-level execvp() function and the
   standard C perror() function. Return nothing if successful, and
   EXIT_FAILURE upon failure. As usual, argc is the command-line
   argument count, and argv contains the command-line arguments. */

int main(int argc, char *argv[])
{
   char *apcArgv[2];

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

   apcArgv[0] = "./hello";
   apcArgv[1] = NULL;

   execvp("./hello", apcArgv);

   perror(argv[0]);
   /* Alternative:
      fprintf(stderr, "%s: %s\n", argv[0], strerror(errno)); */

   exit(EXIT_FAILURE);
}

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

/* Sample executions:

$ gcc217 hello.c -o hello

$ gcc217 testexec.c -o testexec

$ ./testexec
22440 testexec
22440 hello

$ mv hello nothello

$ ./testexec
22454 testexec
./testexec: No such file or directory

$ mv nothello hello

$ ./testexec
22463 testexec
22463 hello

*/
```

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)

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

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>

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

/* Demonstrate the system-level fork() function and the
   standard C fflush() function. Return 0.
   As usual, argc is the command-line argument count, and argv
   contains the command-line arguments. */

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

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

   return 0;
}

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

/* Sample executions:

$ gcc217 testfork.c -o testfork

$ ./testfork
29285 parent
29285 parent and child
29286 parent and child

$ ./testfork
29287 parent
29287 parent and child
29288 parent and child

*/
```

testforkret.c (Page 1 of 2)

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

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>

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

/* Demonstrate how to use the value returned by the system-level
   fork() function. Return 0;
   As usual, argc is the command-line argument count, and argv
   contains the command-line arguments. */

int main(int argc, char *argv[])
{
   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)
      /* This code is executed by the child process only. */
      printf("%d child\n", (int)getpid());
   else
      /* This code is executed by the parent process only. */
      printf("%d parent\n", (int)getpid());

   /* This code is executed by both the parent and child processes. */
   printf("%d parent and child\n", (int)getpid());

   return 0;
}

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

/* Sample executions:

$ gcc217 testforkret.c -o testforkret

$ ./testforkret
29409 parent
29409 parent
29409 parent and child
29410 child
29410 parent and child

$ ./testforkret
29413 parent
29413 parent
29413 parent and child
29414 child
```

testforkret.c (Page 2 of 2)

```
29414 parent and child

*/
```

testforkexit.c (Page 1 of 1)

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

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>

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

/* Demonstrate the common pattern of using the system-level fork()
   function and the standard C exit() function. Return 0.
   As usual, argc is the command-line argument count, and argv
   contains the command-line arguments. */

int main(int argc, char *argv[])
{
   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)
   {
      /* This code is executed by the child process only. */
      printf("%d child\n", (int)getpid());
      exit(0);
   }

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

   return 0;
}

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

/* Sample executions:

$ gcc217 testforkexit.c -o testforkexit

$ ./testforkexit
29595 parent
29595 parent
29596 child

$ ./testforkexit
29600 parent
29600 parent
29601 child

*/
```

testforkloop.c (Page 1 of 2)

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

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>

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

/* Demonstrate context switching among concurrent processes. Return 0.
   As usual, argc is the command-line argument count, and argv
   contains the command-line arguments. */

int main(int argc, char *argv[])
{
   pid_t iPid;
   int iRet;
   int i;

   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)
   {
      /* This code is executed by the child process only. */
      for (i = 0; i < 10 ; i++)
         printf("%d child %d\n", (int)getpid(), i);
      exit(0);
   }

   /* This code is executed by the parent process only. */
   for (i = 0; i < 10; i++)
      printf("%d parent %d\n", (int)getpid(), i);

   return 0;
}

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

/* Sample execution:

$ gcc217 testforkloop.c -o testforkloop

$ ./testforkloop
9857 parent
9857 parent 0
9857 parent 1
9857 parent 2
9857 parent 3
9857 parent 4
9857 parent 5
9857 parent 6
9857 parent 7
```

testforkloop.c (Page 2 of 2)

```
9858 child 0
9857 parent 8
9858 child 1
9857 parent 9
9858 child 2
9858 child 3
9858 child 4
9858 child 5
9858 child 6
9858 child 7
9858 child 8
9858 child 9

*/
```

testforkwait.c (Page 1 of 2)

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

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

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

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

int main(int argc, char *argv[])
{
   pid_t iPid;
   int iRet;
   int i = 0;

   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)
   {
      /* This code is executed by the child process only. */
      for (i = 0; i < 10; i++)
         printf("%d child %d\n", (int)getpid(), i);
      exit(0);
   }

   /* This code is executed by the parent process only. */

   /* Wait for the child process to terminate. */
   iPid = wait(NULL);
   if (iPid == -1) {perror(argv[0]); exit(EXIT_FAILURE); }

   for (i = 0; i < 10; i++)
      printf("%d parent %d\n", (int)getpid(), i);

   return 0;
}

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

/* Sample execution:

$ gcc217 testforkwait.c -o testforkwait

$ ./testforkwait
7275 parent
7276 child 0
```

testforkwait.c (Page 2 of 2)

```
7276 child 1
7276 child 2
7276 child 3
7276 child 4
7276 child 5
7276 child 6
7276 child 7
7276 child 8
7276 child 9
7275 parent 0
7275 parent 1
7275 parent 2
7275 parent 3
7275 parent 4
7275 parent 5
7275 parent 6
7275 parent 7
7275 parent 8
7275 parent 9

*/
```

testforkexecwait.c (Page 1 of 2)

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

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

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

/* Demonstrate the common pattern of using the system-level fork(),
   execvp(), and wait() functions. Return 0.
   As usual, argc is the command-line argument count, and argv
   contains the command-line arguments. */

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

   pid_t iPid;
   int iRet;

   for (;;)
   {
      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)
      {
         /* This code is executed by the child process only. */
         char *apcArgv[2];

         apcArgv[0] = "date";
         apcArgv[1] = NULL;

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

      /* This code is executed by the parent process only. */

      /* Wait for the child process to exit. */
      iPid = wait(NULL);
      if (iPid == -1) {perror(argv[0]); exit(EXIT_FAILURE); }

      /* Pause for SLEEP_SECONDS seconds. */
      sleep(SLEEP_SECONDS);
   }

   /* Should not reach this point. */
}

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

```

testforkexecwait.c (Page 2 of 2)

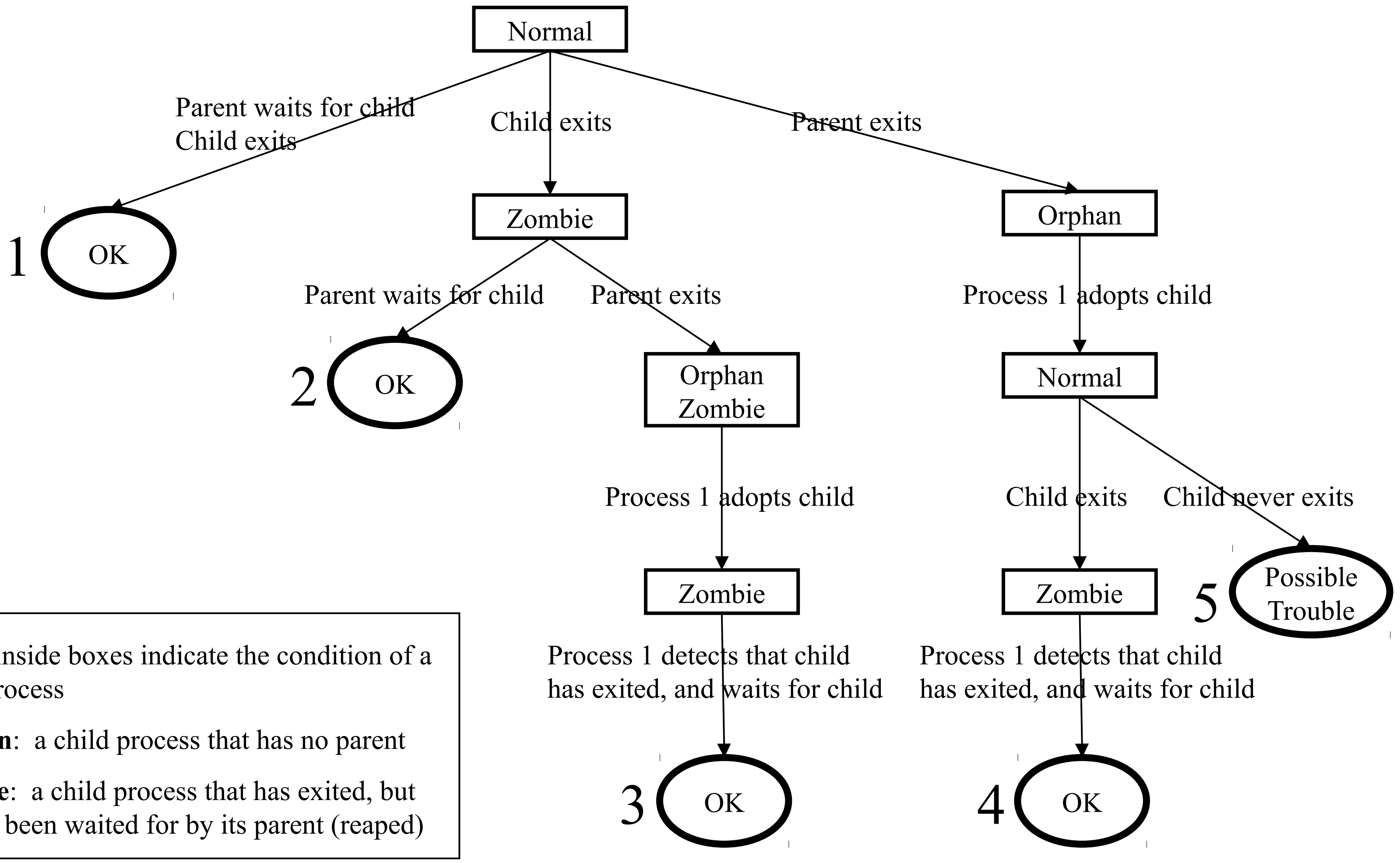
```
/* Sample execution:

$ gcc217 testforkexecwait.c -o testforkexecwait

$ ./testforkexecwait
Wed Apr 24 21:26:13 EDT 2019
Wed Apr 24 21:26:16 EDT 2019
Wed Apr 24 21:26:19 EDT 2019
Wed Apr 24 21:26:22 EDT 2019
Wed Apr 24 21:26:25 EDT 2019
Wed Apr 24 21:26:28 EDT 2019
Wed Apr 24 21:26:31 EDT 2019
Wed Apr 24 21:26:34 EDT 2019
^C

*/
```

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



Terms inside boxes indicate the condition of a child process

Orphan: a child process that has no parent

Zombie: a child process that has exited, but has not been waited for by its parent (reaped)