fork + waitpid + exit

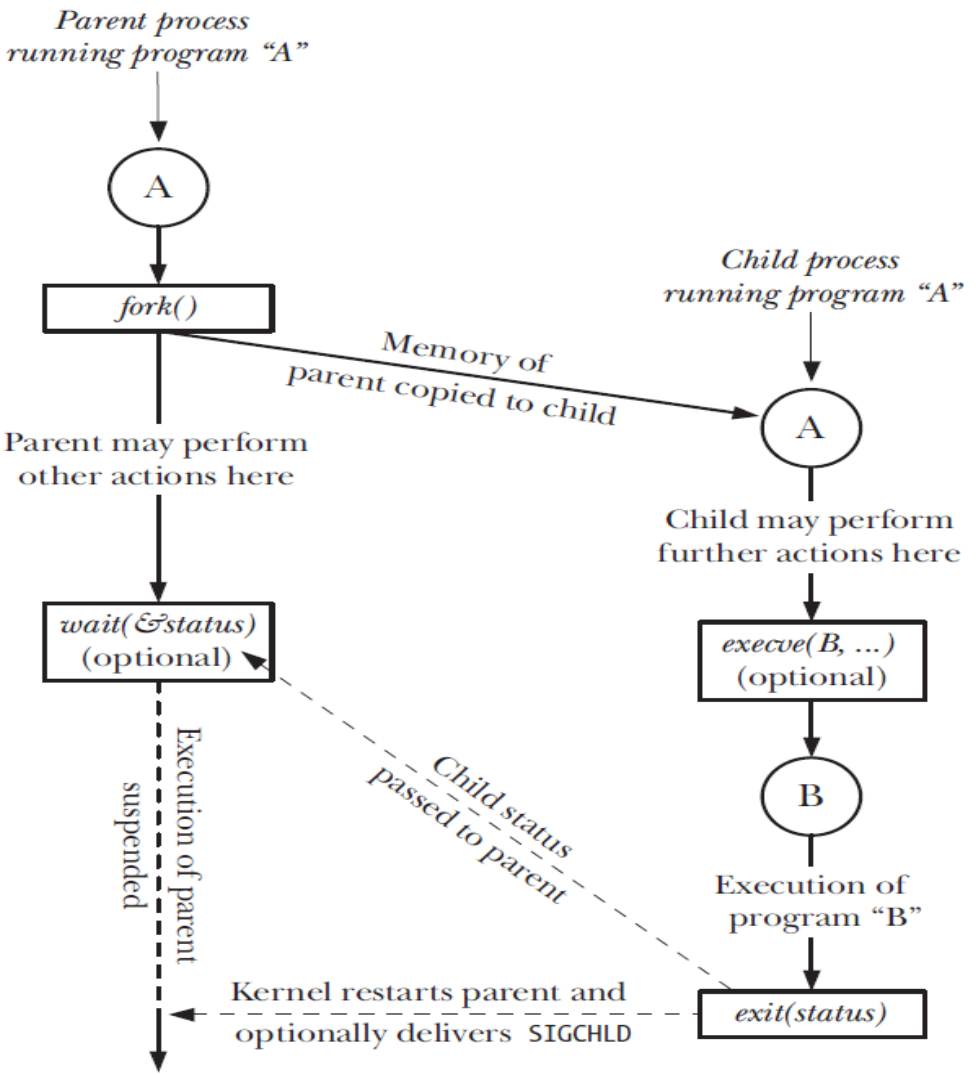


Figure 24-1: Overview of the use of *fork()*, *exit()*, *wait()*, and *execve()*

Programa de un proceso (en GNU/Linux)


```
$ cat prog1.c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int
main(void)
{
  printf("hello world from the process ID %d\n", getpid());
  exit(0);
}

$ gcc prog1.c -o prog1

$ ./prog1
hello world from the process ID 18052

$ ./prog1
hello world from the process ID 18054
```

Programa de un proceso (en macOS)

```
$ cat prog1.c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int
main(void)
{
  printf("hello world from the process ID %d\n", getpid());
  exit(0);
}

$ gcc prog1.c -o prog1

$ ./prog1
hello world from process ID 20189

$ ./prog1
hello world from process ID 20190
```

La vida puede ser muy corta …

Solomon Grundy
(from Mother Goose)

Solomon Grundy,
Born on a Monday,
Christened on Tuesday,
Married on Wednesday,
Took ill on Thursday,
Worse on Friday,
Died on Saturday,
Buried on Sunday.
This is the end
Of Solomon Grundy.

¿Qué procesos viven ahora?

```
$ ps
  PID TTY          TIME CMD
19978 pts/3    00:00:00 bash
20322 pts/3    00:00:00 ps

$ ps u
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
vk        3175  0.0  0.0  41772  2252 pts/0    Ss+  Aug20   0:11
/usr/bin/ssh -o
vk       17915  0.0  0.0  27452  4876 pts/2    Ss+  15:32   0:00 bash
vk       19978  0.0  0.0  27452  4872 pts/3    Ss   17:44   0:00 bash
vk       20127  0.0  0.0  27400  4636 pts/4    Ss   17:57   0:00 bash
vk       20194  0.0  0.0  22244  1964 pts/4    S+   17:59   0:00 man ps
vk       20205  0.0  0.0  17484   992 pts/4    S+   17:59   0:00 pager -s
vk       20323  0.0  0.0  21900  1272 pts/3    R+   18:44   0:00 ps u
```

¿Qué procesos viven ahora?

```
$ ps
  PID TTY           TIME CMD
  224 ttys000    00:00.03 -bash
 1597 ttys001    00:00.01 -bash

$ ps u
USER   PID  %CPU %MEM      VSZ    RSS   TT  STAT STARTED      TIME COMMAND
vk    1597   0.0  0.0  2433436   1224 s001  S+    5:58PM   0:00.01 -bash
vk     224   0.0  0.0  2433436   1260 s000  S     9:34PM   0:00.03 -bash
```

Quiero saber más sobre padres, hijos, nietos


```
$ ps -l x
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY        TIME CMD
...
0 S  1000 18653     1  0  80   0 - 156468 poll_s ?        0:03 gnome-terminal
0 S  1000 18657 18653  0  80   0 -  3705 unix_s ?        0:00 gnome-pty-helper
0 S  1000 18660 18653  0  80   0 -  6370 wait   pts/3    0:00 bash
0 S  1000 18695 18653  0  80   0 -  6366 wait   pts/4    0:00 bash
0 S  1000 18736 18695  0  80   0 -  3027 n_tty_ pts/4    0:00 less
0 R  1000 18933 18660  0  80   0 -  3159 -      pts/3    0:00 ps -l x
```

```
1 -> 18653 -> 18657
          |--> 18660 -> 18933
          |--> 18695 -> 18736
```

```
$ echo $$
18660
```

La variable que contiene el PID del proceso que ejecuta el shell (bash) en este terminal (pts/3)

¿Quién es mi padre?

```
$ cat prog2.c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int
main(void)
{
  printf("hello world from the process ID %d\n", getpid());
  printf("and bye. BTW, my father was the process ID %d\n",
         getppid());
  exit(0);
}
$ gcc prog2.c -o prog2
$ ./prog2
hello world from the process ID 19127
and bye. BTW, my father was the process ID 18695
$ echo $?
0
```

¿Puede decir en qué se especializa el padre del proceso?

¿Qué se queda después de mí?

```
$ cat prog2a.c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int <-- !
main(void)
{
  printf("hello world from the process ID %d\n", getpid());
  printf("and bye. BTW, my father was the process ID %d\n",
         getppid());
  /* exit(0); */
}
$ gcc prog2a.c -o prog2a
$ ./prog2a
hello world from the process ID 19313
and bye. BTW, my father was the process ID 18695
$ echo $?
49 <-- ?
```

Una ayuda del *shell* … (1/3)


```
$ echo


$ echo A
A

$ echo Aaaaa
Aaaaa

$ echo $SHELL
/bin/bash

$ echo 17 > 00000000.txt
$ ls -l 00000000.txt
-rw-r--r-- 1 vk vk 3 mar 29 19:46 00000000.txt

$ cat 00000000.txt
17
```

Una ayuda del *shell* … (2/3)


```
$ man system
```

```
SYSTEM(3)                  Linux Programmer's Manual                  SYSTEM(3)

NAME
       system - execute a shell command

SYNOPSIS
       #include <stdlib.h>

       int system(const char *command);

DESCRIPTION
       system()  executes a command specified in command by calling /bin/sh -c
       command, and returns after the command has been completed.  During exe-
       cution  of the command, SIGCHLD will be blocked, and SIGINT and SIGQUIT
       will be ignored.

RETURN VALUE
       The value returned is -1 on  error  (e.g.,  fork(2)  failed),  and  the
       return  status  of the command otherwise.  This latter return status is
       in the format specified in wait(2).  Thus, the exit code of the command
       will  be  WEXITSTATUS(status).   In case /bin/sh could not be executed,
       the exit status will be that of a command that does exit(127).
```

Sección 3 del manual: Llamadas de la biblioteca (funciones contenidas en las bibliotecas del sistema)

Una ayuda del *shell* … (3/3)


```
$ cat prog3.c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int
main(void)
{
  printf("hello world from the process ID %d\n", getpid());
  system("echo and bye.");
  printf("BTW, my father was the process ID %d\n",
         getppid());
  exit(0);
}
$ gcc prog3.c -o prog3
$ ./prog3
hello world from the process ID 19481
and bye.
BTW, my father was the process ID 18695
```

Soy el hijo del *shell* y el *shell* es mi hijo

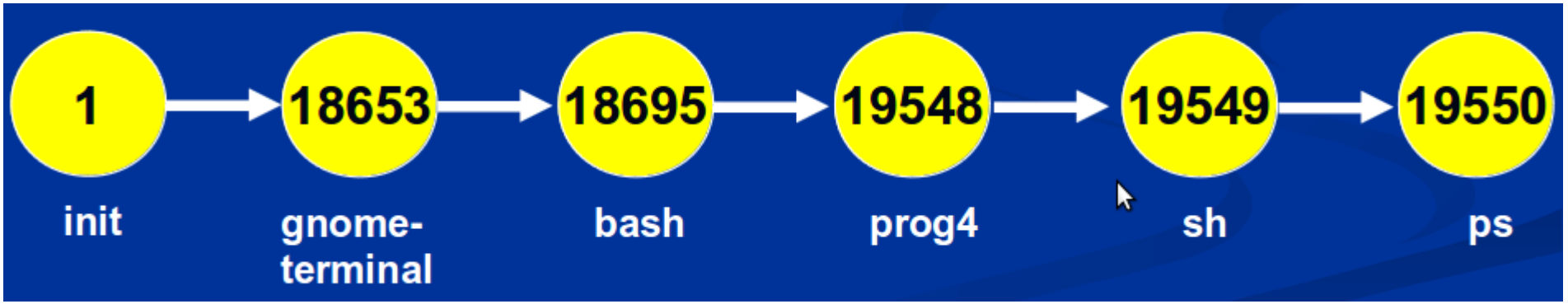

```
$ cat prog4.c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int
main(void)
{
  printf("hello world from the process ID %d\n", getpid());
  system("echo and bye.; ps -l");
  printf("BTW, my father was the process ID %d\n",
         getppid());
  exit(0);
}

$ gcc prog4.c -o prog4
```

Nuestra familia

```
$ ./prog4
hello world from the process ID 19548
and bye.
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 S  1000 18695 18653  0  80   0 -  6394 wait   pts/4    00:00:00 bash
0 S  1000 19082 18695  0  80   0 - 290152 poll_s pts/4   00:00:06 gedit
0 S  1000 19548 18695  0  80   0 -  1048 wait   pts/4    00:00:00 prog4
0 S  1000 19549 19548  0  80   0 -  1110 wait   pts/4    00:00:00 sh
0 R  1000 19550 19549  0  80   0 -  3159 -      pts/4    00:00:00 ps
BTW, my father was the process ID 18695
```



¿No tuvo problemas mi hijo?


```
int
main(void)
{
  int status;

  printf("hello world from the process ID %d\n", getpid());
  status = system("echo and bye.; ps -l");
  printf("status (of ps) = %d\n", status);
  printf("BTW, my father was the process ID %d\n", getppid());
  exit(0);
}
```

```
$ ./prog5
hello world from the process ID 20396
and bye.
F S   UID   PID  PPID  PGRP     SZ         RECV TTY  TIME CMD
...
status (of ps) = 0
BTW, my father was the process ID 18695
```

Lo que devuelve system (1/4)

```
...
status = system("echo and bye.; foo");
printf("status (of foo) = %d\n", status);
...
```

```
$ ./prog5a
hello world from the process ID 20491
and bye.
sh: 1: foo: not found
status (of foo) = 32512
BTW, my father was the process ID 18695
```

Lo que devuelve system (2/4)


```
...
status = system("echo and bye.; foo; exit 44");
printf("status = %d\n", status);
...
```

```
$ ./prog5b
hello world from the process ID 20671
and bye.
sh: 1: foo: not found
status = 11264
BTW, my father was the process ID 18695
```

Lo que devuelve system (3/4)

```
...
status = system("echo and bye.; foo; exit 44");
printf("status = %d\n", WEXITSTATUS(status));
...
```

```
$ ./prog5c
hello world from the process ID 20760
and bye.
sh: 1: foo: not found
status = 44
BTW, my father was the process ID 18695
```

```
$ man waitpid
...
       WEXITSTATUS(status)
              returns the exit status of the child. This consists of the
              least significant 8 bits of the status argument that the child
              specified in a call to exit(3) or _exit(2) or as the argument
              for a return statement in main(). This macro should be employed
              only if WIFEXITED returned true.
```

Lo que devuelve system (4/4)


```
   ...
   status = system("echo and bye.; foo");
   printf("status = %d\n", WEXITSTATUS(status));
   ...

$ ./prog5d
hello world from the process ID 20837
and bye.
sh: 1: foo: not found
status = 127
BTW, my father was the process ID 18695

# Errors that are detected by the shell, such as a syntax
# error, will cause the shell to exit with a non-zero exit
# status.
```

La *única* forma de crear un proceso en el sistema UNIX es mediante la llamada al sistema *fork*. (Con exclusión de algunos procesos especiales que se crean por el kernel.)

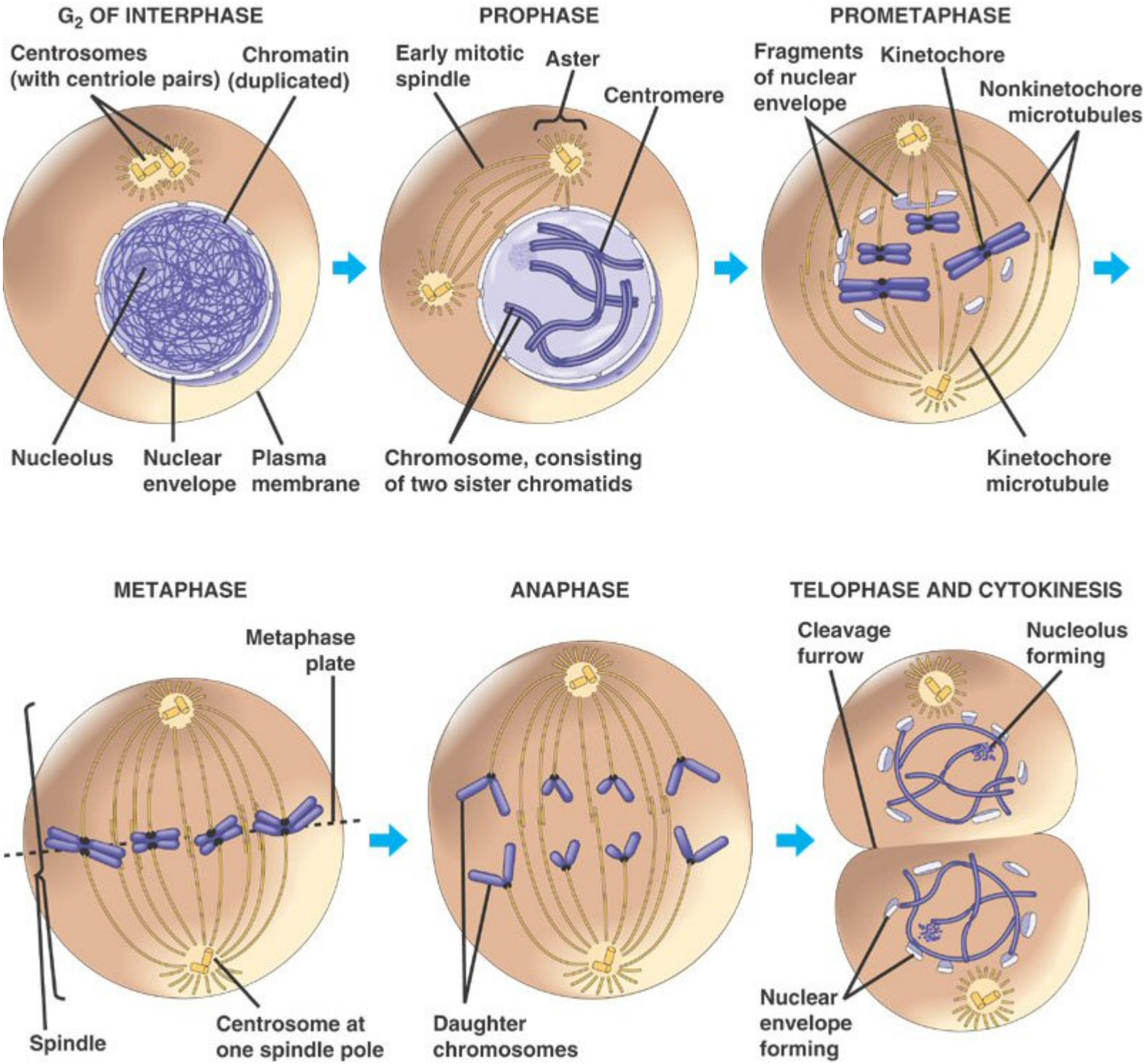
```
#include <sys/types.h>
#include <unistd.h>

pid_t fork(void);
```

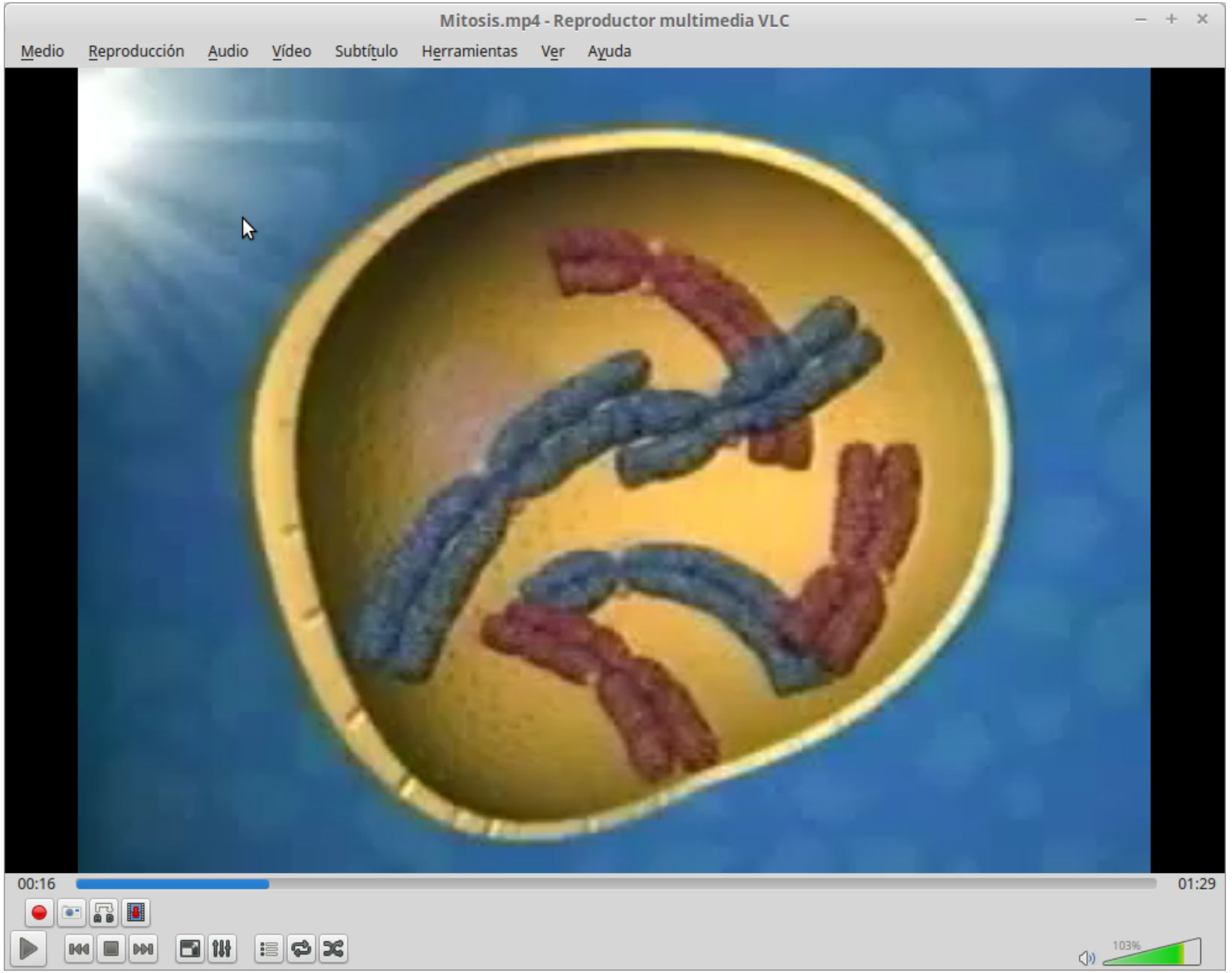
Devuelve:

0 en el hijo,
PID del hijo en el padre,
-1 en caso de error

Mitosis


G2 OF INTERPHASE
Centrosomes (with centriole pairs)
Chromatin (duplicated)
Nucleolus
Nuclear envelope
Plasma membrane
PROPHASE
Early mitotic spindle
Aster
Centromere
Chromosome, consisting of two sister chromatids
PROMETAPHASE
Fragments of nuclear envelope
Kinetochore
Nonkinetochore microtubules
Kinetochore microtubule
METAPHASE
Metaphase plate
Spindle
Centrosome at one spindle pole
ANAPHASE
Daughter chromosomes
TELOPHASE AND CYTOKINESIS
Cleavage furrow
Nucleolus forming
Nuclear envelope forming

Mitosis


Mitosis.mp4 - Reproductor multimedia VLC
Medio
Reproducción
Audio
Vídeo
Subtítulo
Herramientas
Ver
Ayuda
00:16
01:29
103%

El proceso que invoca a *fork* se llama *proceso padre* y el proceso creado es el *proceso hijo*.

La llamada a *fork* hace que el proceso actual se duplique.

A la salida de *fork*, los dos procesos tienen una copia idéntica del contexto del nivel de usuario excepto el valor de *PID*.

Como hay dos procesos, la llamada *fork* devuelve 2 valores, uno para cada uno de los procesos.

Al proceso hijo se le devuelve 0, y al proceso padre se le devuelve el PID de su hijo nuevo.

Ambos procesos siguen su ejecución con la instrucción que sigue a la llamada *fork*.

```
int main (int argc, char **argv)
{
  int retval;

  printf("Definitivamente es el padre\n");
  retval = fork();
  printf("¿Qué proceso imprime esto?\n");
  return (EXIT_SUCCESS);
}
```

```
Definitivamente es el padre
¿Qué proceso imprime esto?
¿Qué proceso imprime esto?
```

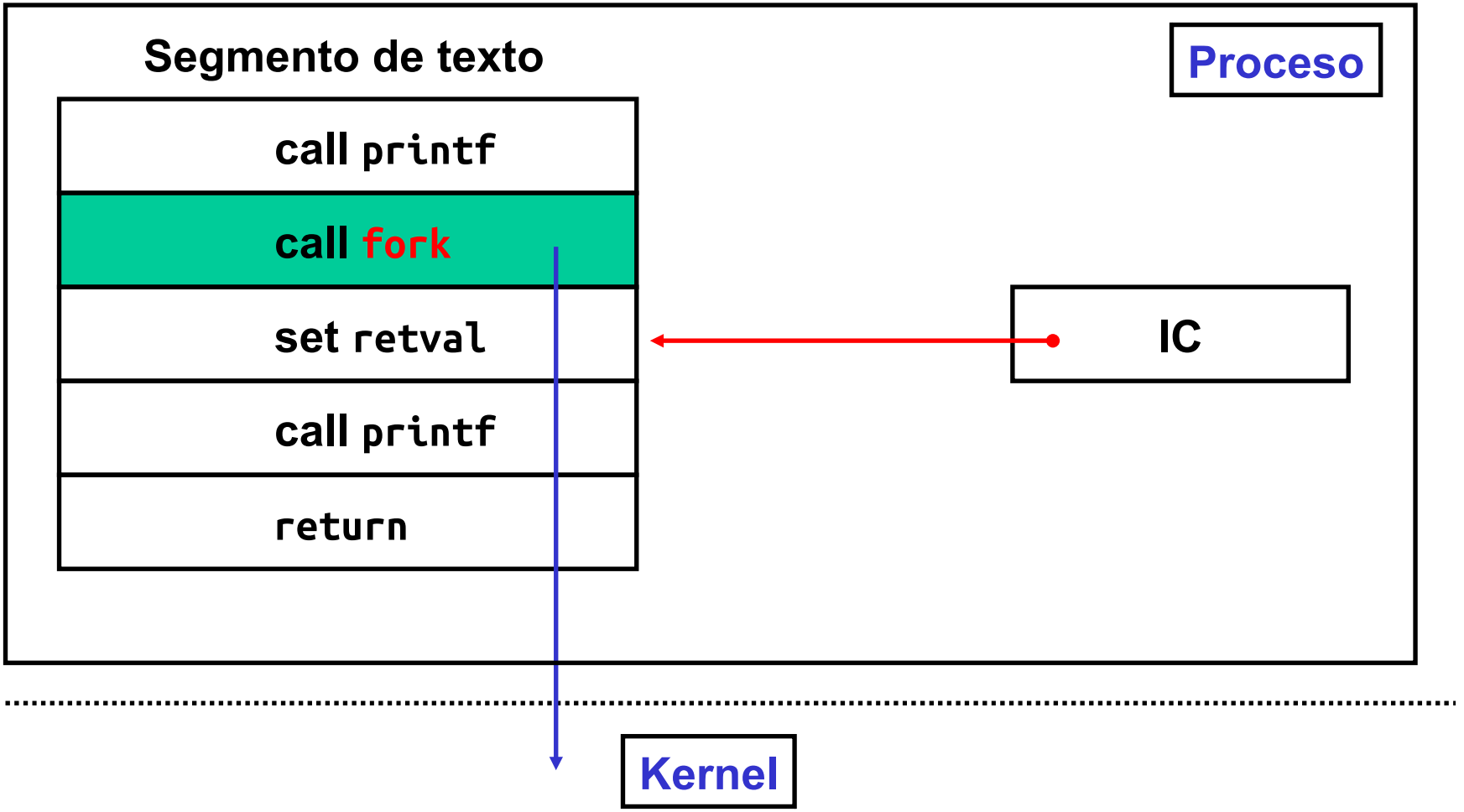
Un fork … y ya somos dos

```
  pid_t pid;

  printf("pid = %d, ppid = %d\n", getpid(), getppid());
  fflush(stdout);
  if ((pid=fork()) == -1) {
    perror("fork failed");
    exit(1);
  }
  printf("pid = %d, ppid = %d\n", getpid(), getppid());
  printf("that's all\n");
  exit(0);
```

```
$ ./prog6
pid = 21442, ppid = 18695
pid = 21442, ppid = 18695
that's all
pid = 21443, ppid = 21442
that's all
```

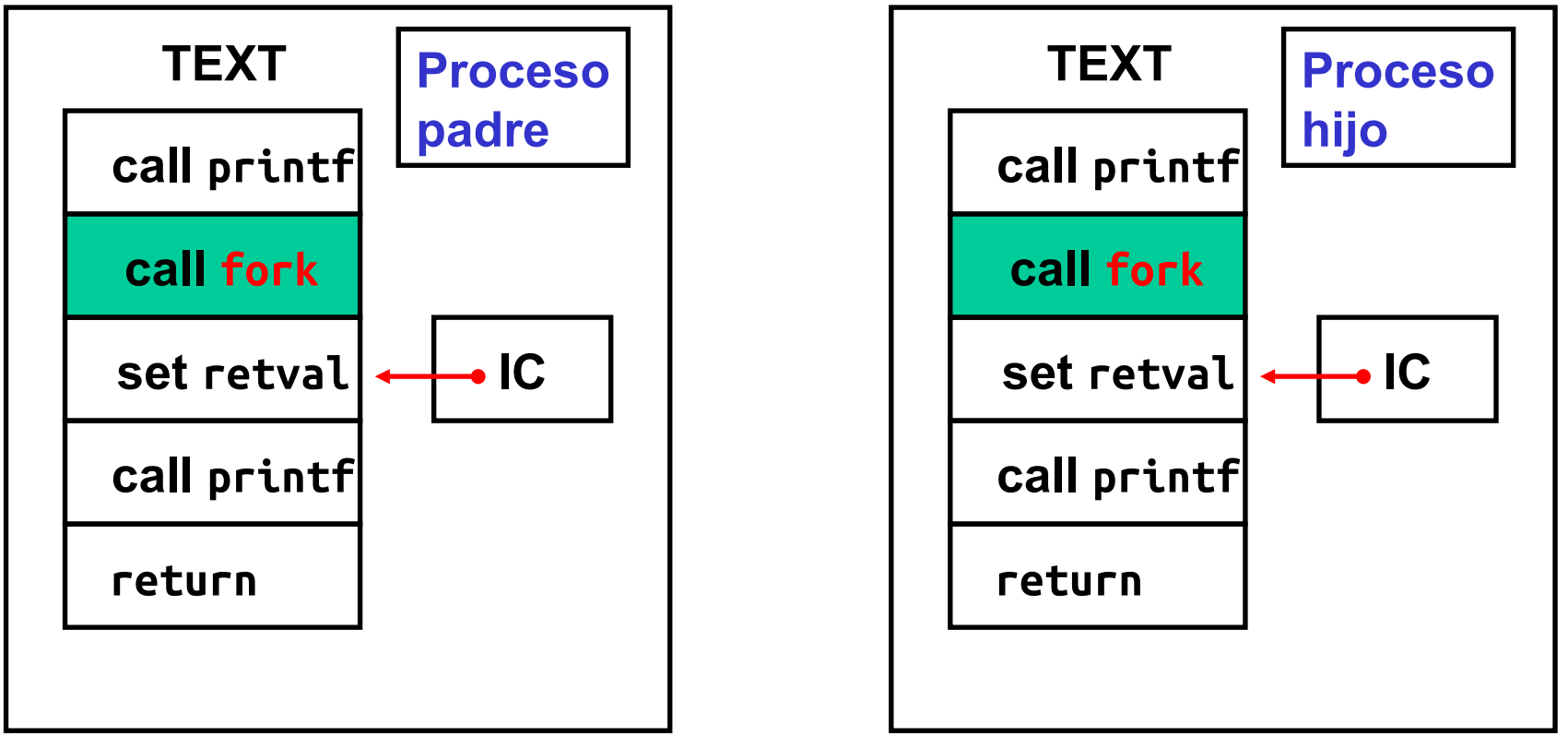
Proceso hace llamada al sistema `fork`



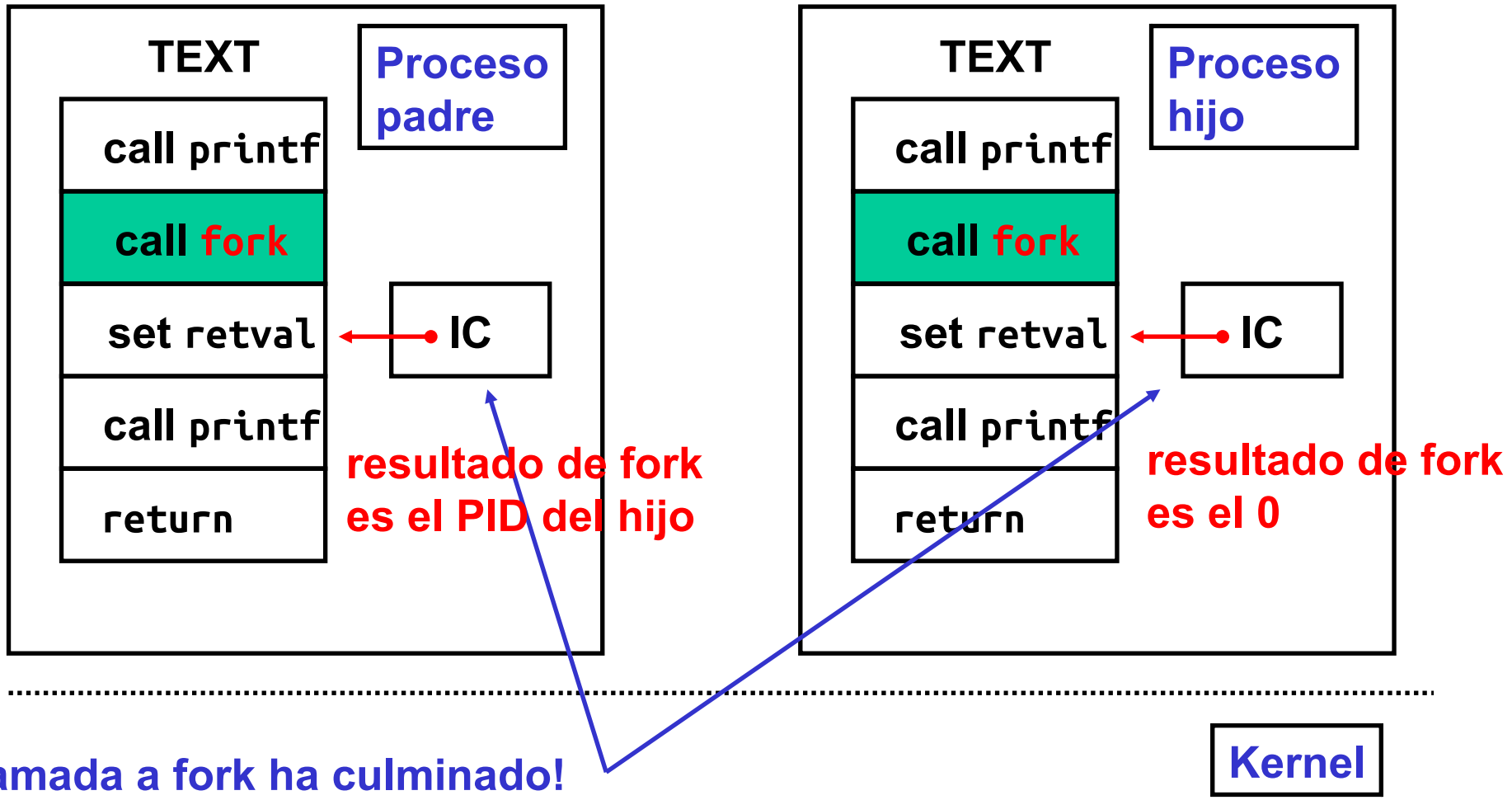
Con la llamada fork, el kernel realiza las siguientes operaciones:

1. **Busca una entrada libre en la *tabla de procesos* y la reserva para el proceso hijo.**
2. **Asigna un identificador de proceso para el proceso hijo.**
3. **Realiza una copia del contexto del nivel de usuario del proceso padre para el proceso hijo. Las secciones que deban ser compartidas, como el código o las zonas de memoria compartida, no se copian, sino que se incrementan los contadores que indican cuántos procesos comparten estas zonas.**
4. **Las tablas de control de archivos locales al proceso, como puede ser la tabla de descriptores de archivos, también se copian del proceso padre al proceso hijo, ya que forman parte del contexto del nivel de usuario. En las tablas globales del kernel, tabla de archivos y tabla de *inodes*, se incrementan los contadores que indican cuántos procesos tienen abiertos esos archivos.**
5. **Retorna al proceso padre el PID del proceso hijo, y al proceso hijo le devuelve el valor 0.**

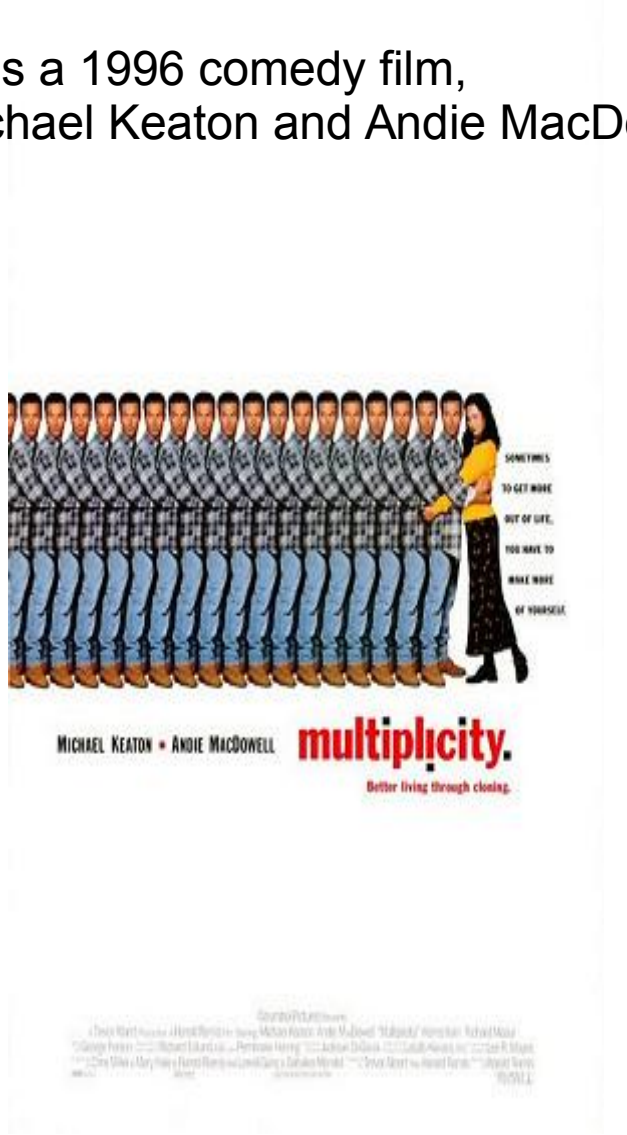
Proceso padre y proceso hijo (1/2)



Proceso padre y proceso hijo (2/2)



Multiplicity is a 1996 comedy film,
starring Michael Keaton and Andie MacDowell.



The Process class in Python (1/3)


```
$ cat process_1.py
#!/usr/bin/env python3
#   Filename: process_1.py
#   Multiprocessing - Python v3.3.0 documentation
#   http://docs.python.org/2/library/multiprocessing.html
#   16.6.1.1. The Process class

from multiprocessing import Process

def f(name):
    print('hello', name)

if __name__ == '__main__':
    p = Process(target=f, args=('bob',))
    p.start()
    p.join()

$ ./process_1.py
hello bob
```

The Process class in Python (2/3)


```
$ cat process_2.py
#!/usr/bin/env python3
#   Multiprocessing - Python v3.3.0 documentation
#   http://docs.python.org/2/library/multiprocessing.html
#   16.6.1.1. The Process class

from multiprocessing import Process
import os

def info(title):
    print(title)
    print('module name: ', __name__)
    if hasattr(os, 'getppid'):
        print('parent process: ', os.getppid())
    print('process id: ', os.getpid())

def f(name):
    info('function f')
    print('hello', name)
```

The Process class in Python (3/3)

```
...
if __name__ == '__main__':
    info('main line')
    p = Process(target=f, args=('bob',))
    p.start()
    p.join()

$ echo $$      # Process ID
5926
$ ./process_2.py
main line
module name:  __main__
parent process:  5926
process id:  25439
function f
module name:  __main__
parent process:  25439
process id:  25440
hello bob
```

Cuando un proceso pierde a su padre … (1/2)


```
$ cat prog7.c
...
int
main(void)
{
  pid_t pid, ppid;

  printf("pid = %d, ppid = %d\n", getpid(), getppid());
  fflush(stdout);
  if ((pid=fork() == -1) {
    perror("fork failed"); exit(1);
  }
  if (!pid) sleep(1); /* child's dream */
  printf("pid = %d, ppid = %d\n", getpid(), (ppid=getppid()));
  if (ppid == 1) printf("I have a new father - init\n");
  printf("that's all\n");
  exit(0);
}
```

Cuando un proceso pierde a su padre … (2/2)


```
$ gcc prog7.c -o prog7

$ ./prog7
pid = 21756, ppid = 18695
pid = 21756, ppid = 18695
that's all
$ pid = 21757, ppid = 1
I have a new father - init
that's all

$
```

El proceso padre terminó,
lo esperaba el *shell*,
ahora el shell está listo para recibir
la siguiente orden y por eso
produce el *prompt*.

Creando un *daemon* … (1/2)


```
$ cat prog7a.c
#include <stdlib.h>
#include <unistd.h>

int
main(void)
{
  /* a nonstandard function, don't use it! */
  if (daemon(0,0) == -1) {
    perror("daemon function failed");
    exit(1);
  }

  while(1);
}
```

Creando un *daemon* … (2/2)


```
$ gcc prog7a.c -o prog7a

$ ./prog7

$ ps -l -C prog7a
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
1 R  1000  3398     1 99  80   0 -  1047 -      ?        00:00:13 prog7a

$ kill -KILL 3398

$ ps -l -C prog7a
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD

$
```

Formando una cadena de los procesos … (1/2)


```
$ cat prog8.c
...
int
main(void)
{
  int i, n=4;
  pid_t pid, ppid;

  for (i=0; i<n; i++) {
    if ((pid=fork() == -1) { perror("fork failed"); exit(1); }
    if (pid) break;
  }
  printf("pid = %d, ppid = %d\n", getpid(), (ppid=getppid()));
  sleep(5);
  printf("that's all\n");
  exit(0);
}
```

Formando una cadena de los procesos … (2/2)


```
$ gcc prog8.c -o prog8

$ ./prog8
pid = 21869, ppid = 18695
pid = 21870, ppid = 21869
pid = 21871, ppid = 21870
pid = 21872, ppid = 21871
pid = 21873, ppid = 21872
that's all
that's all
that's all
that's all
that's all
```

18695 → 21869 → 21870 → 21871 → 21872 → 21873

Todos procesos ejecutan el mismo programa


```
$ ./prog8 &
[2] 21927
$ pid = 21927, ppid = 18695
pid = 21928, ppid = 21927
pid = 21929, ppid = 21928
pid = 21930, ppid = 21929
pid = 21931, ppid = 21930
ps -l
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 S  1000 18695 18653  0  80   0 -  6394 wait   pts/4    00:00:00 bash
0 S  1000 21927 18695  0  80   0 -  1048 hrtime pts/4    00:00:00 prog8
1 S  1000 21928 21927  0  80   0 -  1048 hrtime pts/4    00:00:00 prog8
1 S  1000 21929 21928  0  80   0 -  1048 hrtime pts/4    00:00:00 prog8
1 S  1000 21930 21929  0  80   0 -  1048 hrtime pts/4    00:00:00 prog8
1 S  1000 21931 21930  0  80   0 -  1048 hrtime pts/4    00:00:00 prog8
0 R  1000 21932 18695  0  80   0 -  3159 -      pts/4    00:00:00 ps
$ that's all
that's all
that's all
that's all
that's all
```

Formando una cadena de los procesos pero … (1/2)


```
$ cat prog9.c
...
int
main(void)
{
  int i, n=4;
  pid_t pid, ppid;

  for (i=0; i<n; i++) {
    if ((pid=fork() == -1) { perror("fork failed"); exit(1); }
    if (pid) break;
  }
  printf("pid = %d, ppid = %d\n", getpid(), (ppid=getppid()));
  if (pid) sleep(5);
  printf("that's all\n");
  exit(0);
}
```

Formando una cadena de los procesos pero … (2/2)


```
$ gcc prog9.c -o prog9

$ ./prog9
pid = 22088, ppid = 18695
pid = 22089, ppid = 22088
pid = 22090, ppid = 22089
pid = 22091, ppid = 22090
pid = 22092, ppid = 22091
that's all
ps -l
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 S  1000 18695 18653  0  80   0 -  6394 wait   pts/4    00:00:00 bash
0 S  1000 22088 18695  0  80   0 -  1048 hrtime pts/4    00:00:00 prog9
1 S  1000 22089 22088  0  80   0 -  1048 hrtime pts/4    00:00:00 prog9
1 S  1000 22090 22089  0  80   0 -  1048 hrtime pts/4    00:00:00 prog9
1 S  1000 22091 22090  0  80   0 -  1048 hrtime pts/4    00:00:00 prog9
1 Z  1000 22092 22091  0  80   0 -     0 exit   pts/4    00:00:00 pro <defunct>
0 R  1000 22093 18695  0  80   0 -  3159 -      pts/4    00:00:00 ps
$ that's all
that's all
that's all
that's all
```

Zombie process

Tomando responsabilidades paternas … (1/2)


```
$ cat prog10.c
...
#include <sys/wait.h>

int
main(void)
{
  int i, n=4;
  pid_t childpid, mypid, ppid;

  for (i=0; i<n; i++) {
    if ((childpid=fork() == -1) {perror("fork failed");exit(1);}
    if (childpid) break;
  }
  printf("pid = %d, ppid = %d\n", (mypid=getpid()),
                                  (ppid=getppid()));
  if (childpid) waitpid(childpid, NULL, 0);
  printf("%d: that's all\n", mypid);
  exit(0);
}
```

Tomando responsabilidades paternas … (2/2)


```
$ gcc prog10.c -o prog10

$ ./prog10
pid = 22568, ppid = 18695
pid = 22569, ppid = 22568
pid = 22570, ppid = 22569
pid = 22571, ppid = 22570
pid = 22572, ppid = 22571
22572: that's all
22571: that's all
22570: that's all
22569: that's all
22568: that's all
$
```

Formando un “abanico” de procesos … (1/2)

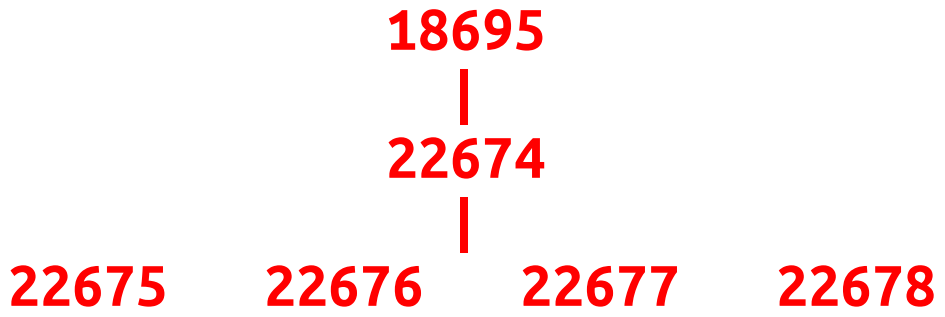

```
$ cat prog11.c
...
int
main(void)
{
  int i, n=4;
  pid_t childpid, mypid, ppid;

  for (i=0; i<n; i++) {
    if ((childpid=fork() == -1) {perror("fork failed");exit(1);}
    if (!childpid) break;
  }
  printf("pid = %d, ppid = %d\n", (mypid=getpid()),
                                  (ppid=getppid()));
  if (childpid) for (i=0; i<n; i++) waitpid(-1, NULL, 0);
  else sleep(1);
  printf("%d: that’s all\n", mypid);
  exit(0);
}
```

Formando un “abanico” de procesos … (2/2)

```
$ gcc prog11.c -o prog11

$ ./prog11
pid = 22675, ppid = 22674
pid = 22676, ppid = 22674
pid = 22674, ppid = 18695
pid = 22677, ppid = 22674
pid = 22678, ppid = 22674
22675: that's all
22676: that's all
22678: that's all
22677: that's all
22674: that's all
$
```



Formando un árbol de procesos … (1/2)

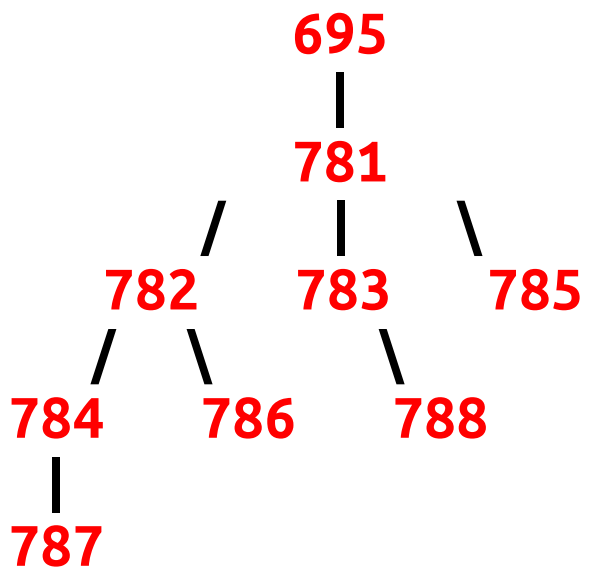

```
$ cat prog12.c
...
int
main(void)
{
  int i, n=3;
  pid_t childpid, mypid, ppid; /* no es el momento para getppid() */

  for (i=0; i<n; i++)
    if ((childpid=fork() == -1) {perror("fork failed");exit(1);}

  printf("pid = %d, ppid = %d\n", (mypid=getpid()),
                                  (ppid=getppid()));
  if (childpid) while (waitpid(-1, NULL, 0) != -1);
  else sleep(1);
  printf("%d: that's all\n", mypid);
  exit(0);
}
```

Formando un árbol de procesos … (2/2)

```
$ ./prog12
pid = 22781, ppid = 18695
pid = 22782, ppid = 22781
pid = 22784, ppid = 22782
pid = 22787, ppid = 22784
pid = 22786, ppid = 22782
pid = 22783, ppid = 22781
pid = 22788, ppid = 22783
pid = 22785, ppid = 22781
22786: that's all
22787: that's all
22784: that's all
22785: that's all
22782: that's all
22788: that's all
22783: that's all
22781: that's all
$
```

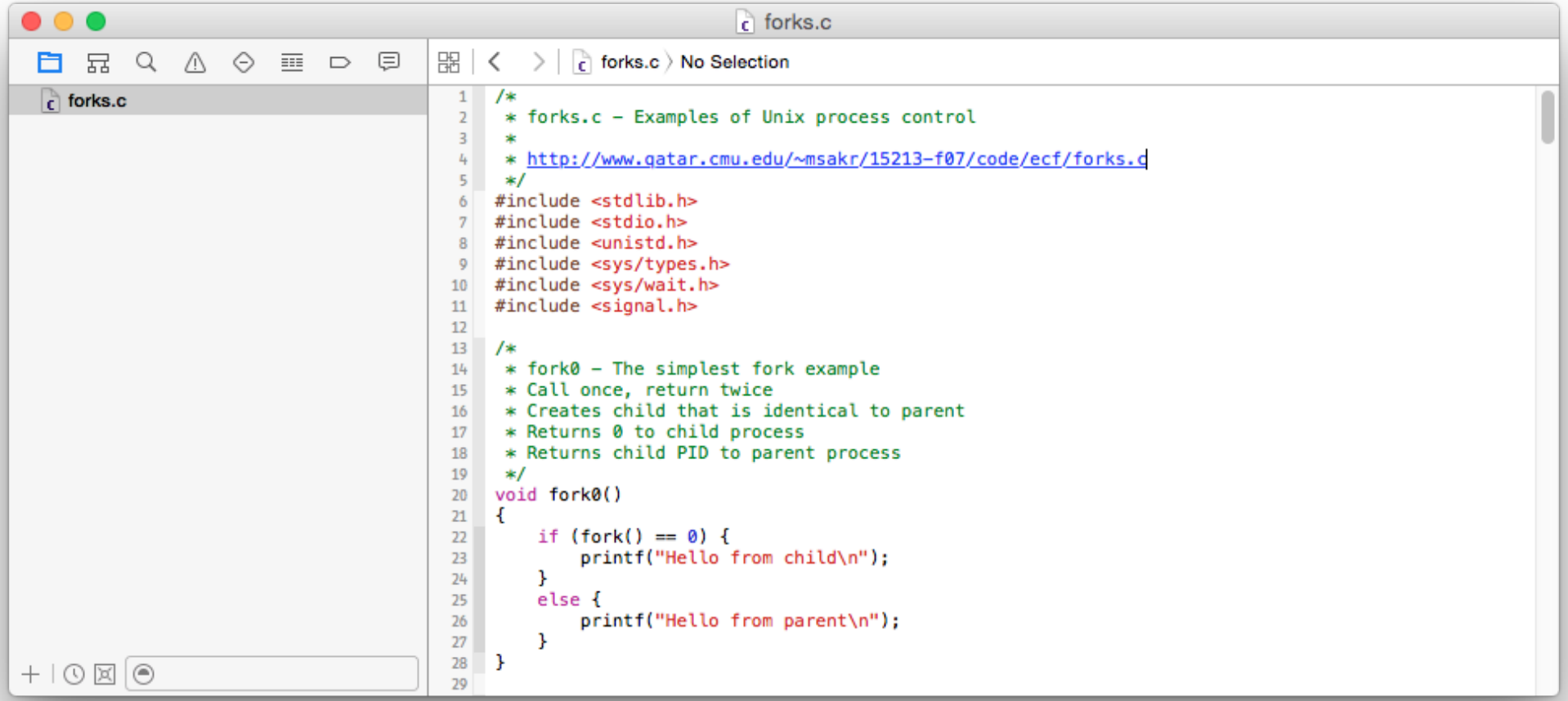


Otro ejemplo de fork

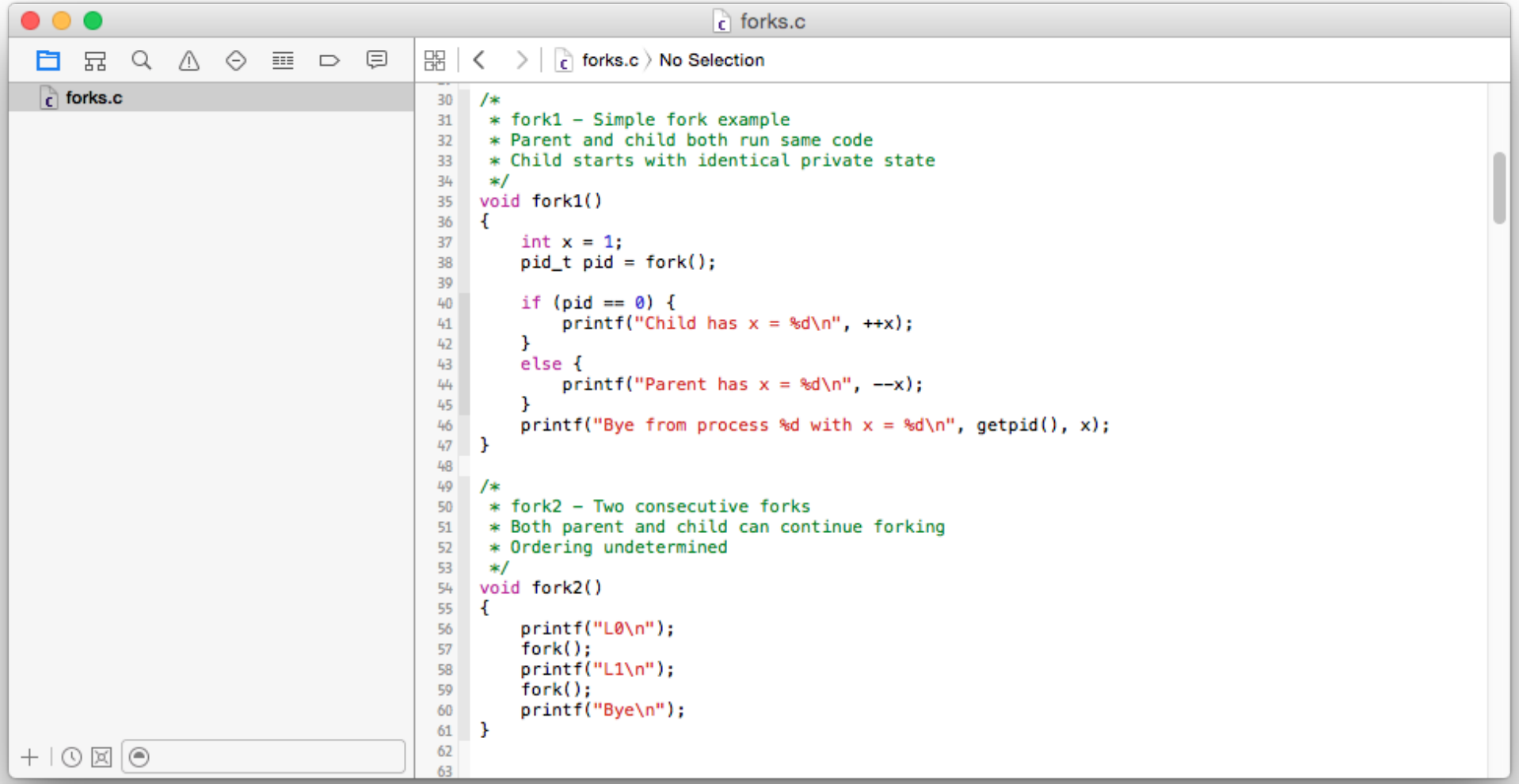
```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>

if((childpid=fork()) == 0) {
  fprintf(stderr, "Soy el hijo, ID = %ld\n",
          (long)getpid());
  /* el código del hijo va aquí */
} else if (childpid > 0) {
  fprintf(stderr, "Soy el padre, ID = %ld\n",
          (long)getpid());
  /* el código del padre va aquí */
}
```

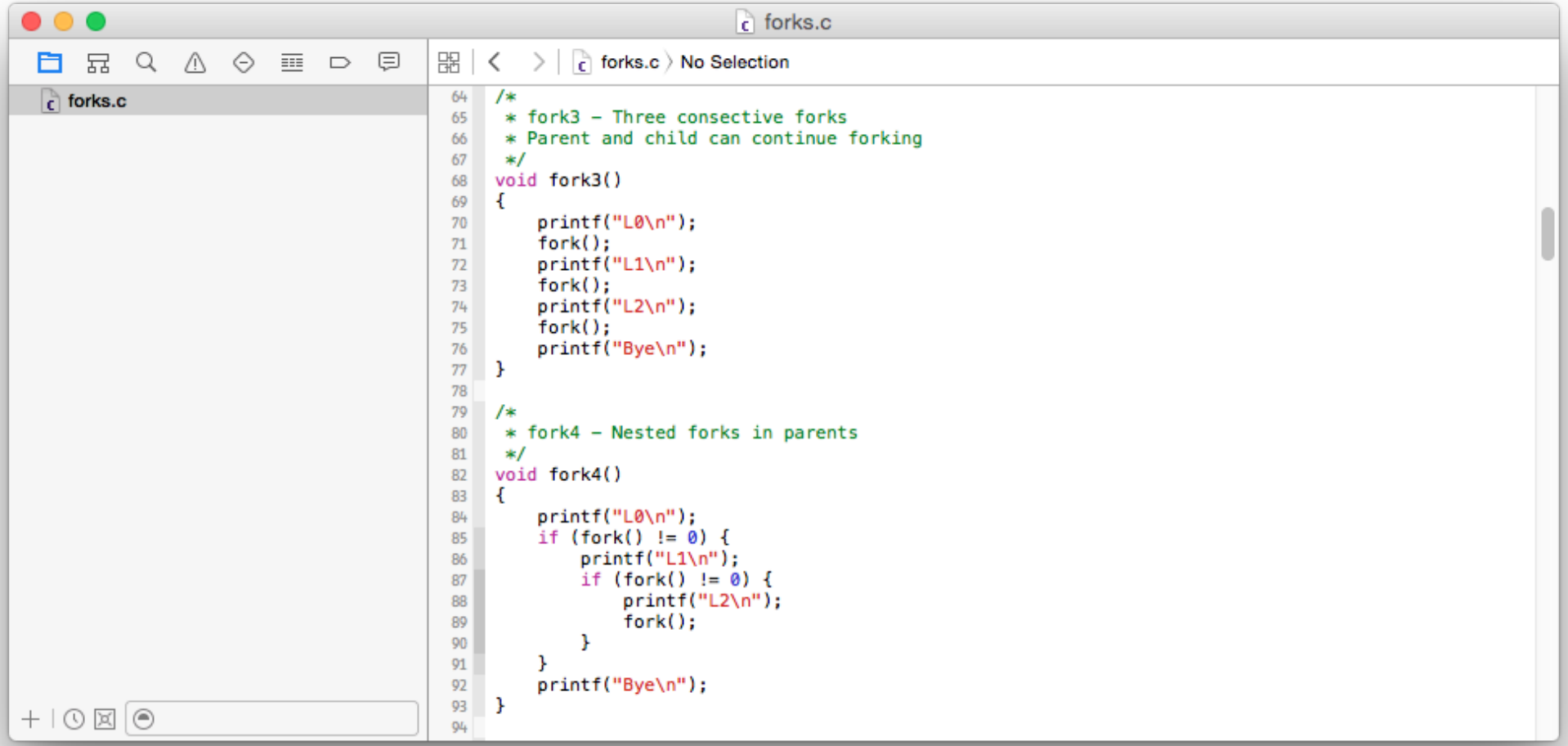
http://www.qatar.cmu.edu/~msakr/15213-f07/code/ecf/forks.c



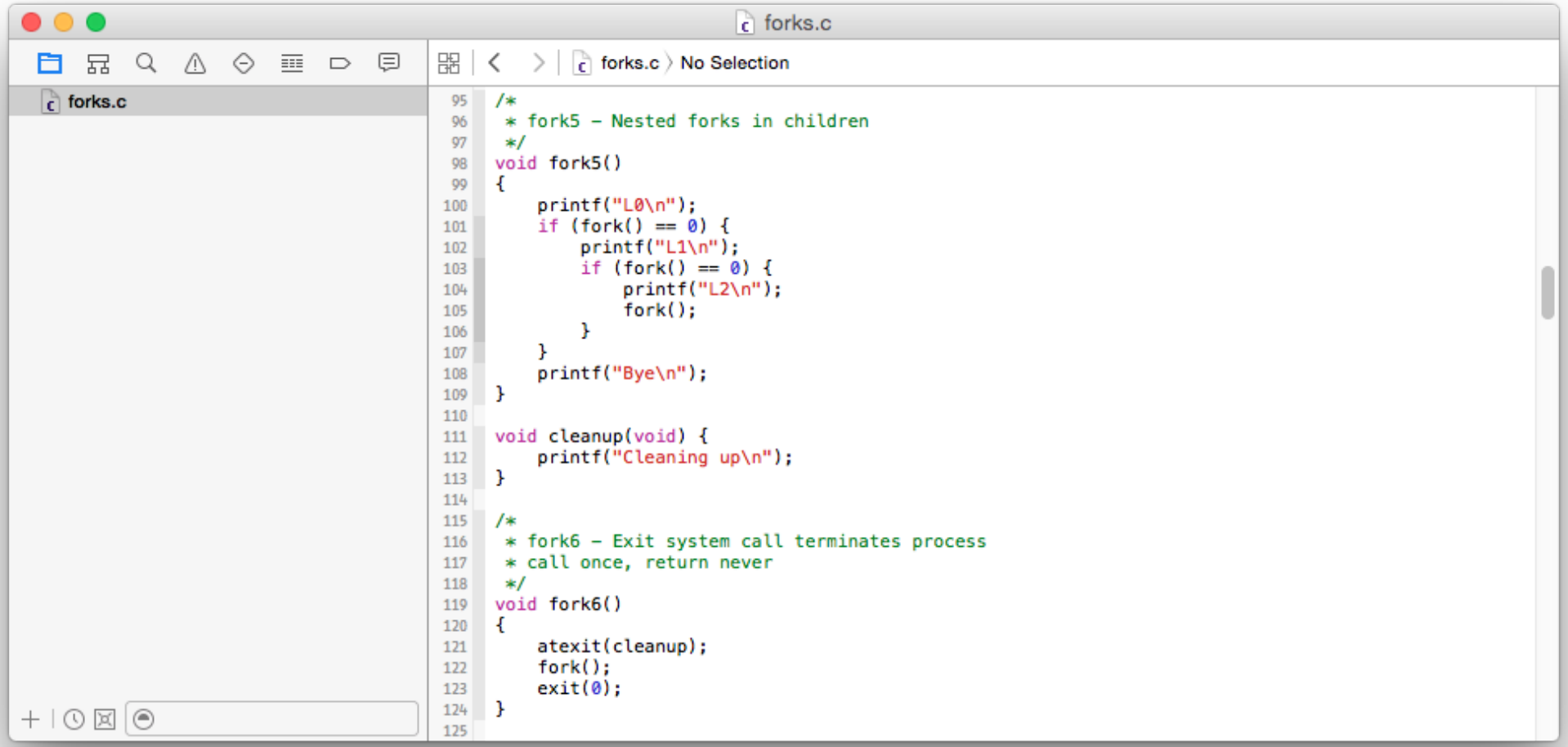
http://www.qatar.cmu.edu/~msakr/15213-f07/code/ecf/forks.c



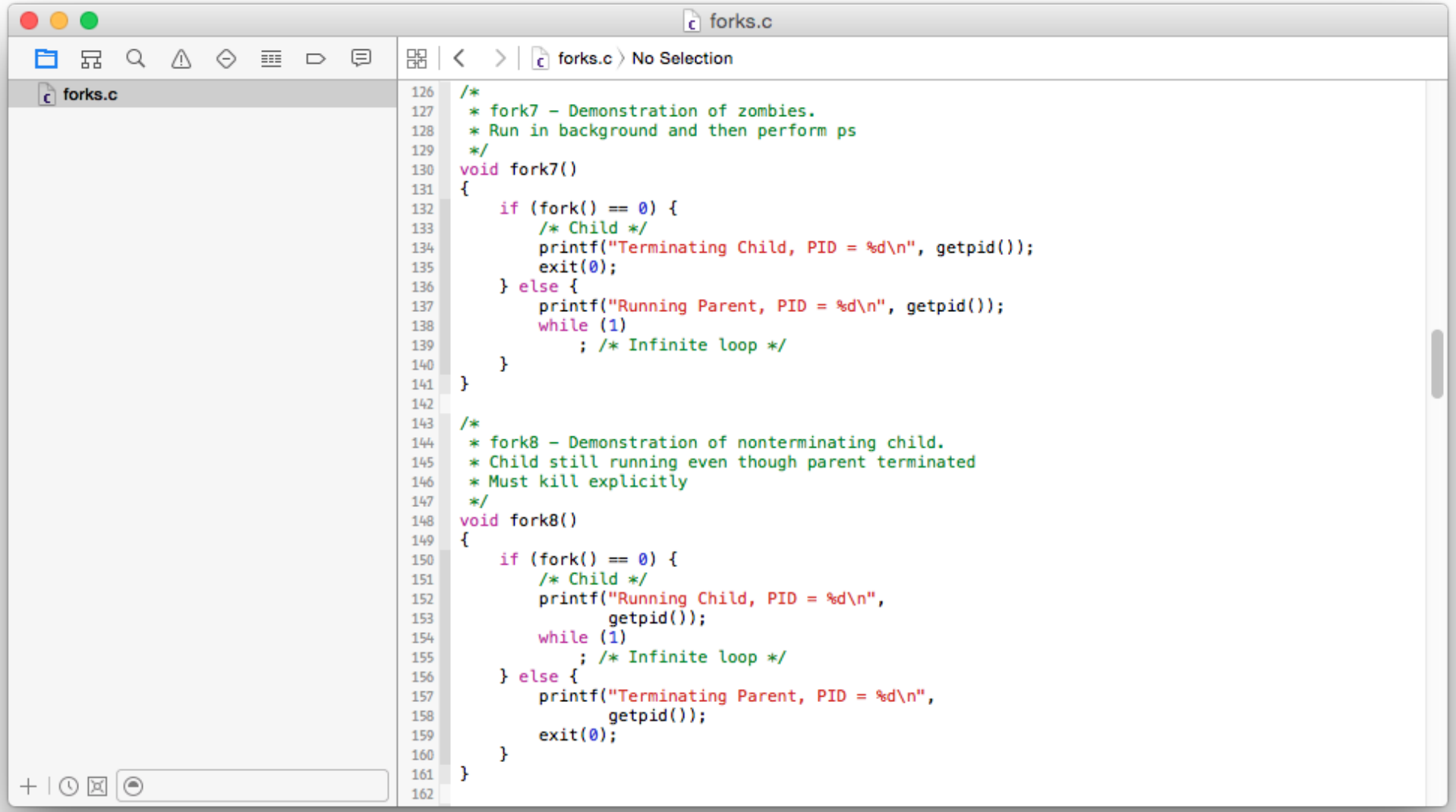
http://www.qatar.cmu.edu/~msakr/15213-f07/code/ecf/forks.c



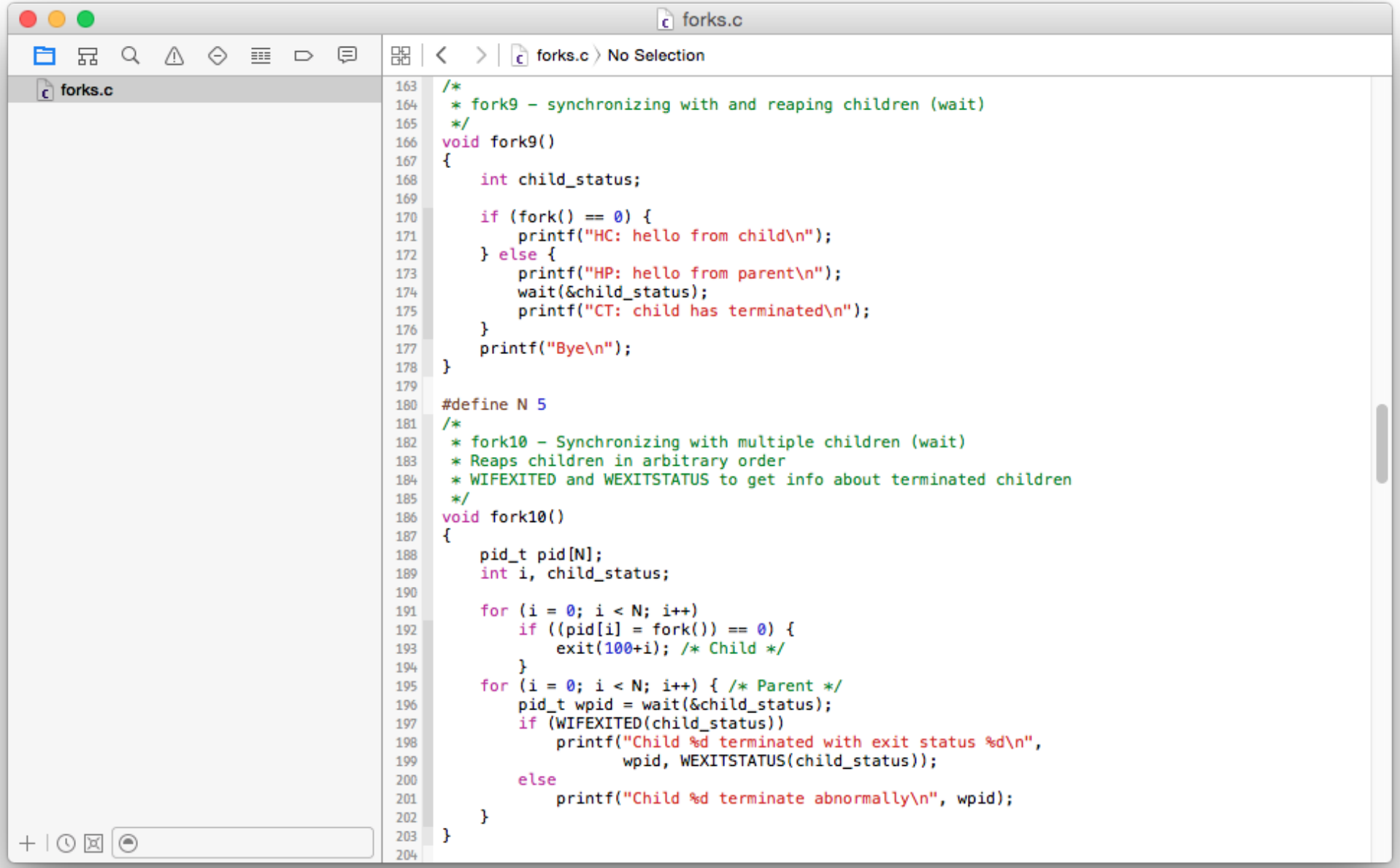
http://www.qatar.cmu.edu/~msakr/15213-f07/code/ecf/forks.c



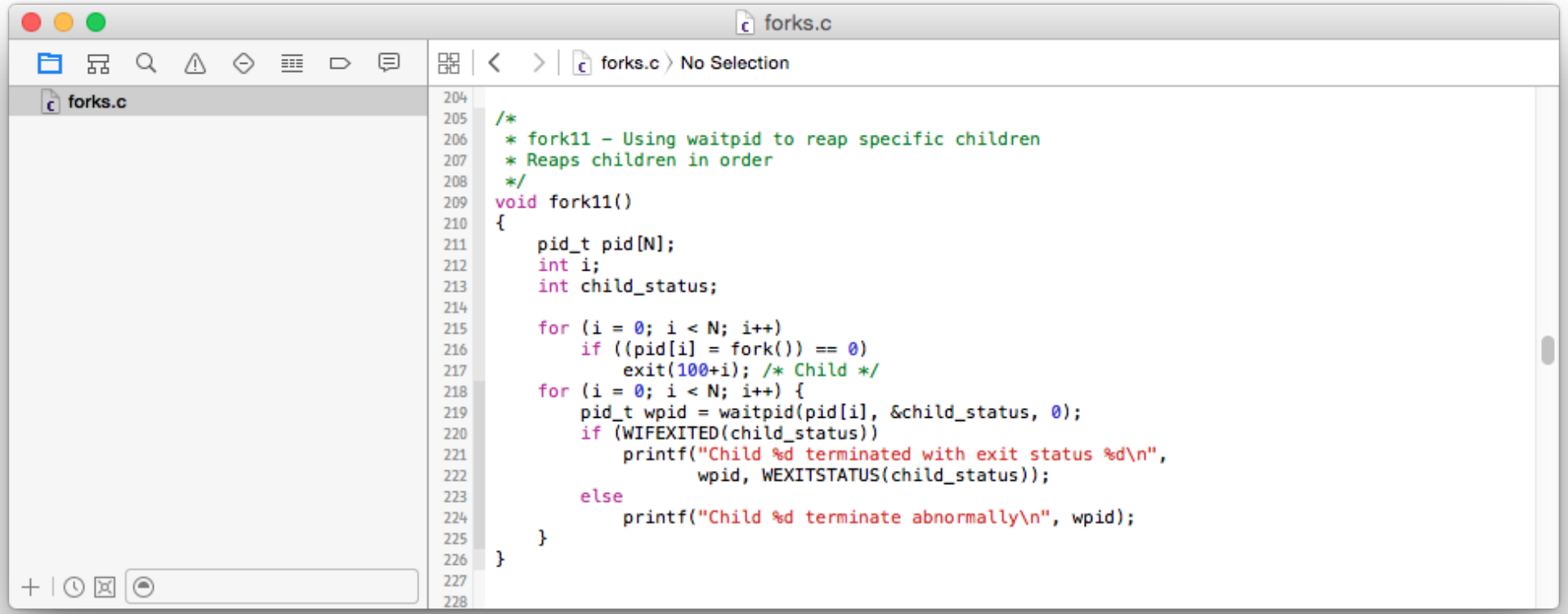
http://www.qatar.cmu.edu/~msakr/15213-f07/code/ecf/forks.c



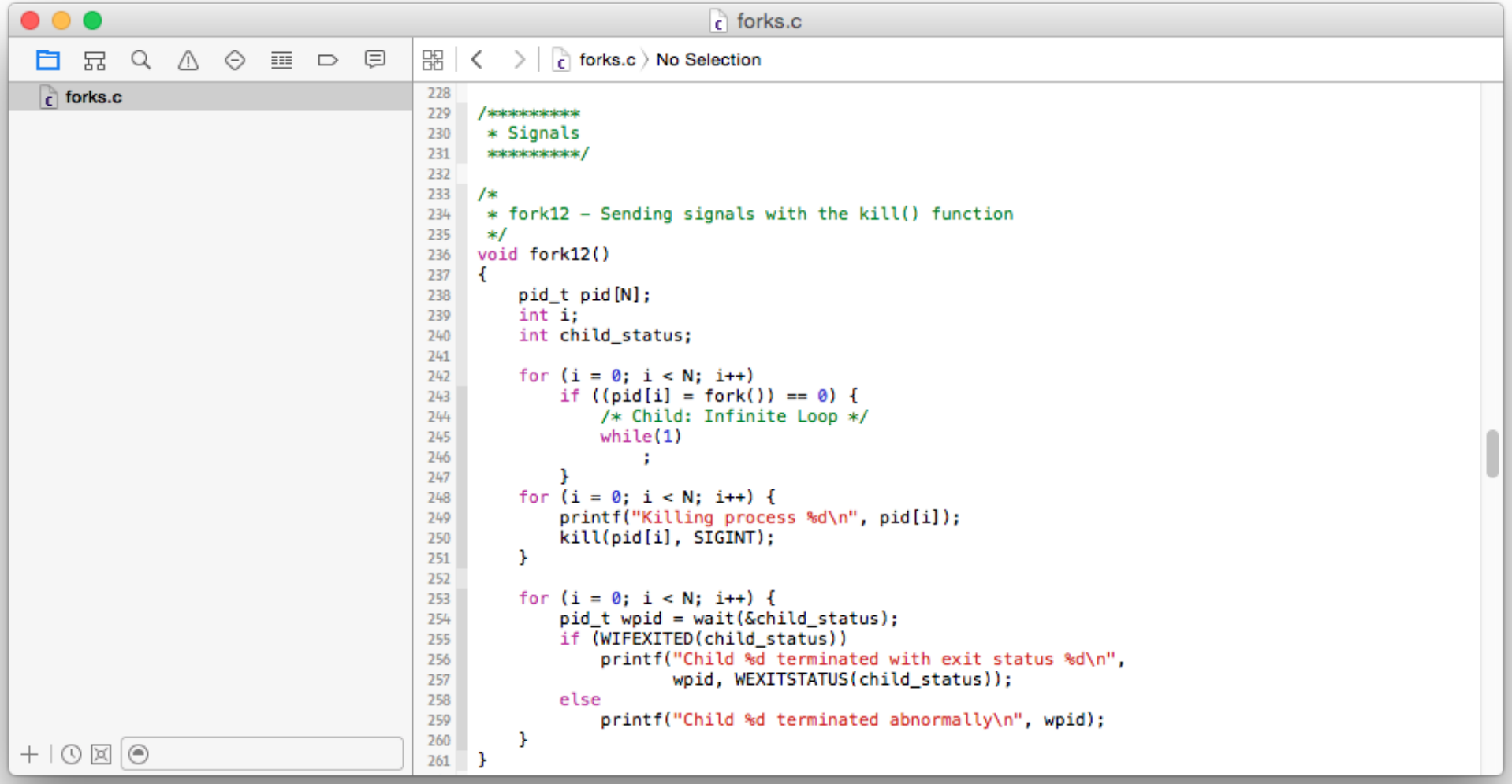
http://www.qatar.cmu.edu/~msakr/15213-f07/code/ecf/forks.c



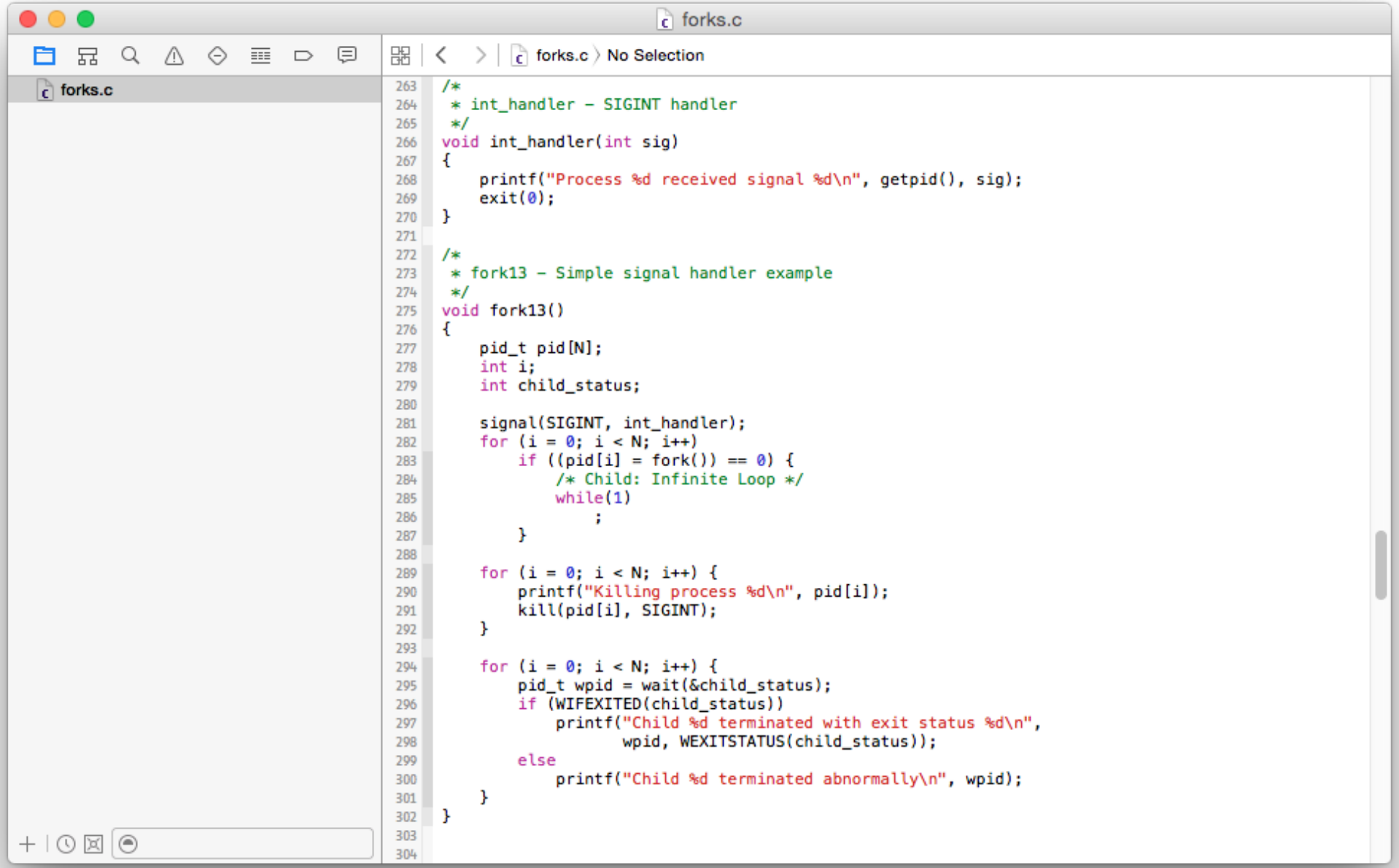
http://www.qatar.cmu.edu/~msakr/15213-f07/code/ecf/forks.c



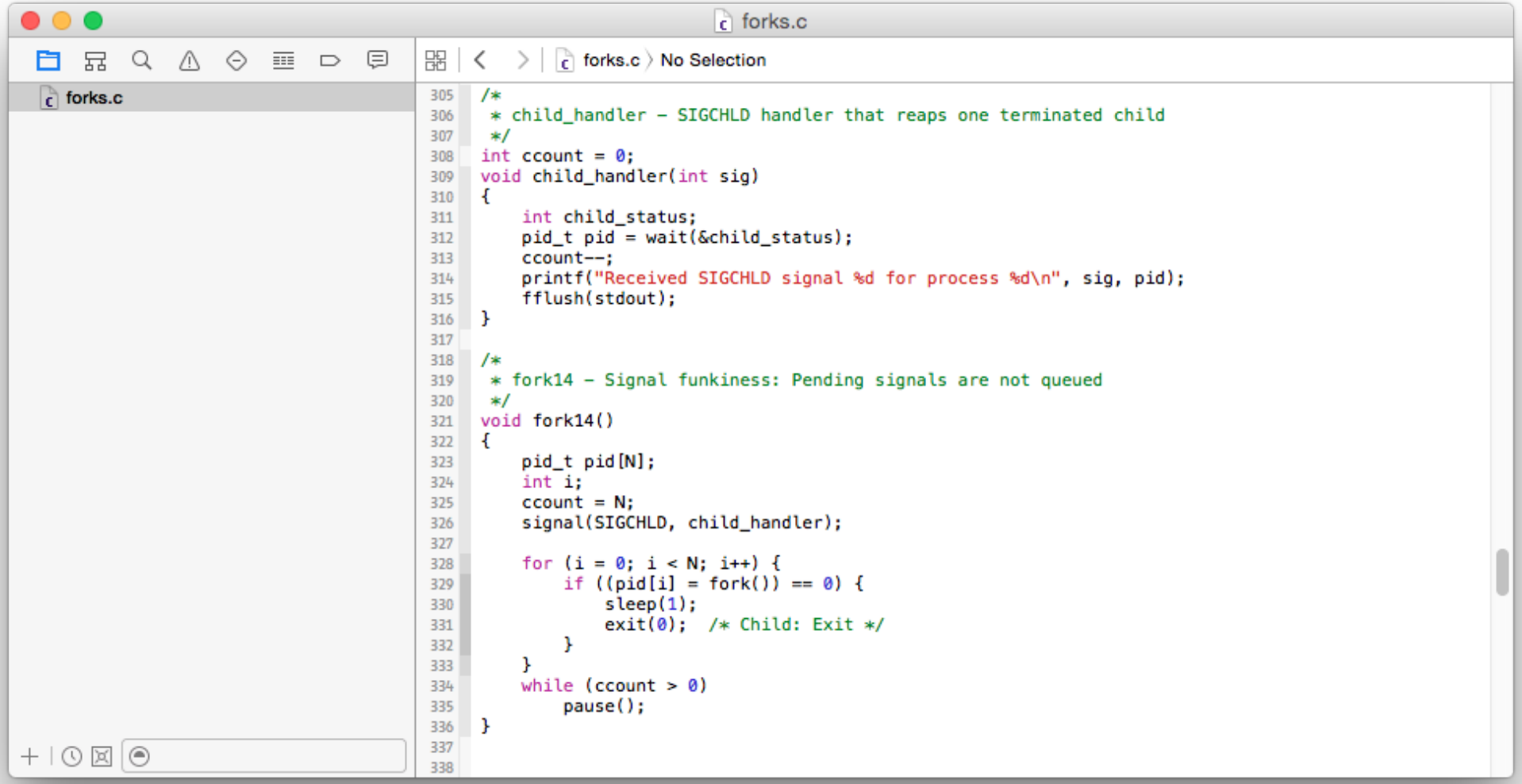
http://www.qatar.cmu.edu/~msakr/15213-f07/code/ecf/forks.c



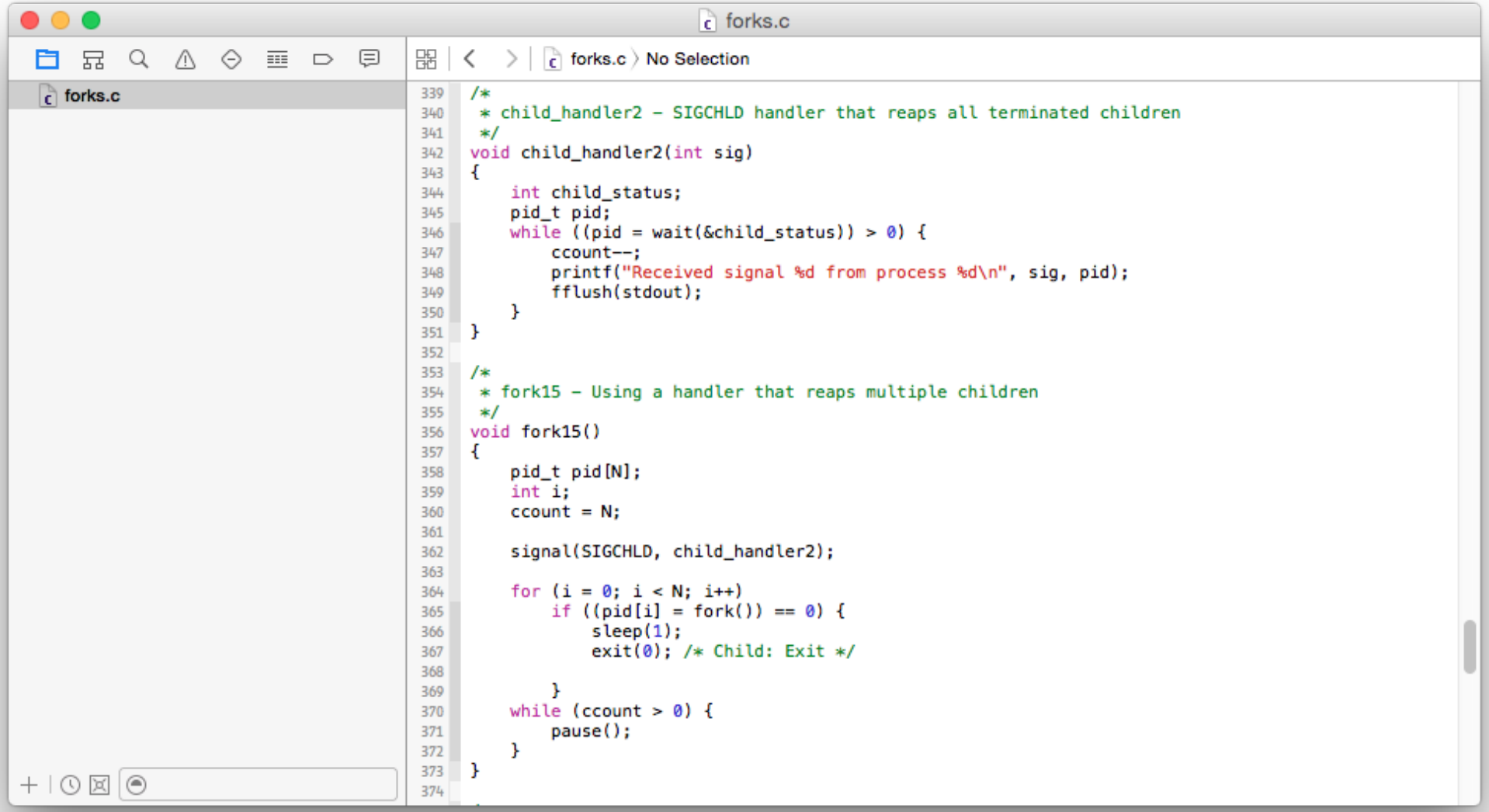
http://www.qatar.cmu.edu/~msakr/15213-f07/code/ecf/forks.c



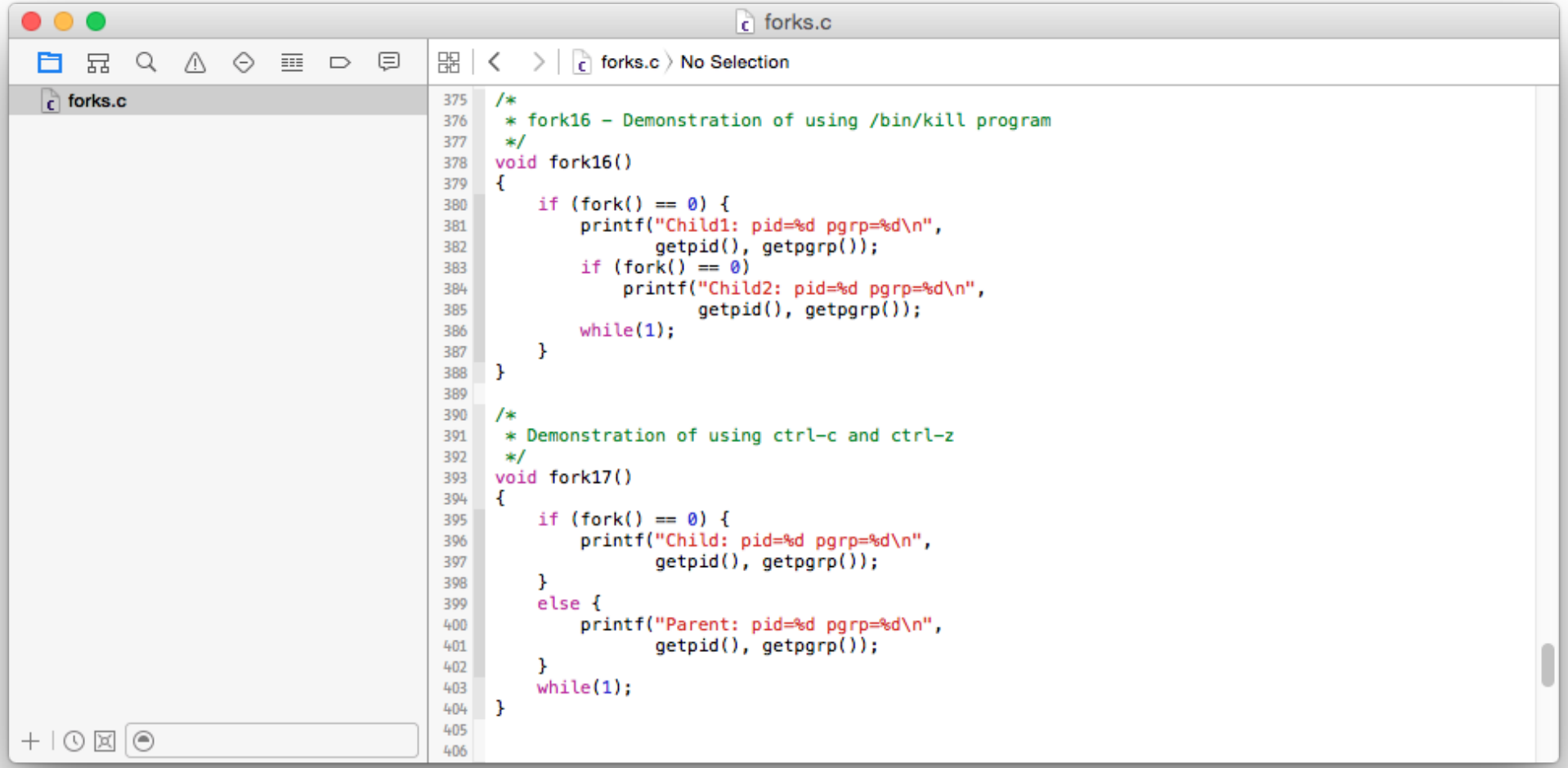
http://www.qatar.cmu.edu/~msakr/15213-f07/code/ecf/forks.c



http://www.qatar.cmu.edu/~msakr/15213-f07/code/ecf/forks.c



http://www.qatar.cmu.edu/~msakr/15213-f07/code/ecf/forks.c



http://www.qatar.cmu.edu/~msakr/15213-f07/code/ecf/forks.c

forks.c

forks.c > No Selection

```
int main(int argc, char *argv[])
{
    int option = 0;
    if (argc > 1)
        option = atoi(argv[1]);
    switch(option) {
        case 0: fork0();
            break;
        case 1: fork1();
            break;
        case 2: fork2();
            break;
        case 3: fork3();
            break;
        case 4: fork4();
            break;
        case 5: fork5();
            break;
        case 6: fork6();
            break;
        case 7: fork7();
            break;
        case 8: fork8();
            break;
        case 9: fork9();
            break;
        case 10: fork10();
            break;
        case 11: fork11();
            break;
        case 12: fork12();
            break;
        case 13: fork13();
            break;
        case 14: fork14();
            break;
        case 15: fork15();
            break;
        case 16: fork16();
            break;
        case 17: fork17();
            break;
        default:
            printf("Unknown option %d\n", option);
            break;
    }
    return 0;
}
```

```
$ ./forks
Hello from parent
Hello from child

$ ./forks 1
Parent has x = 0
Bye from process 975 with x = 0
Child has x = 2
Bye from process 976 with x = 2

$ ./forks 2
L0
L1
Bye
L1
Bye
Bye
Bye
```

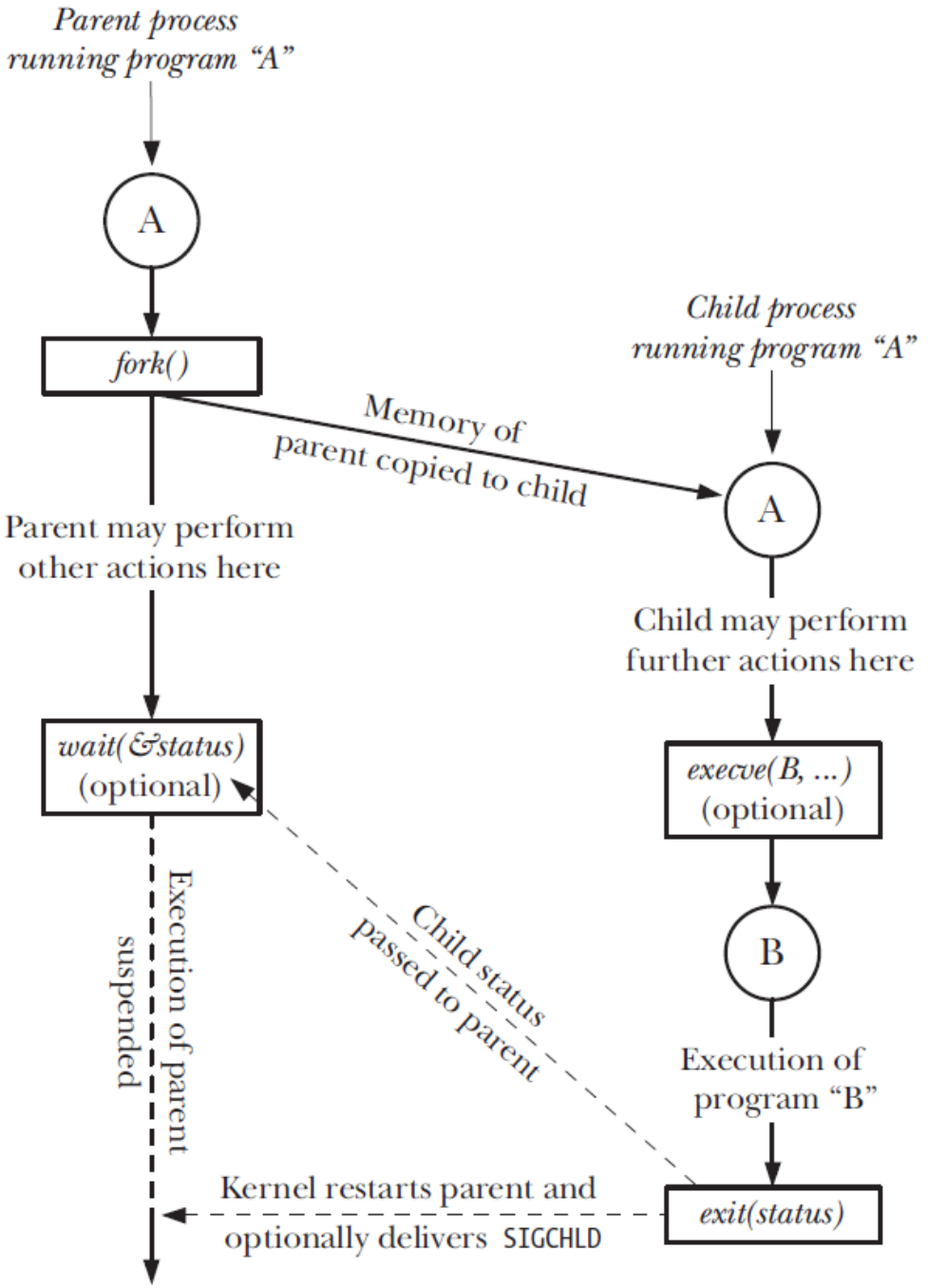


Figure 24-1: Overview of the use of *fork()*, *exit()*, *wait()*, and *execve()*

The Linux Programming Interface
by Michael Kerrisk