```c
#define _XOPEN_SOURCE 500
#define __EXTENSIONS__
#define _POSIX_C_SOURCE 199506L

#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <signal.h>
#include <sys/mman.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <sys/stat.h>
#include <errno.h>
#include <stdarg.h>
#include <string.h>
#include <stdlib.h>
#include <stdint.h>
#include <time.h>

#ifndef MAP_ANONYMOUS
#define MAP_ANONYMOUS MAP_ANON
#endif

/* ================== Constantes ================== */

#define NUM_SLOTS_PIDS 20
#define TAM_BUF_MSJ    512
#define FICHERO_PIDS   "pids.bin"


#define RES_SABRINA 1
#define RES_JILL    2
#define RES_KELLY   4


static const char *ruta_ejecutable_programa = NULL;

/* ================== Mascara global de arranque ================== */

static sigset_t mascara_senales_bloqueo;

static void bloquear_senales_de_arranque(void)
{
    sigemptyset(&mascara_senales_bloqueo);
    sigaddset(&mascara_senales_bloqueo, SIGUSR1);
    sigaddset(&mascara_senales_bloqueo, SIGUSR2);

    if (sigprocmask(SIG_BLOCK, &mascara_senales_bloqueo, NULL) < 0) {
        perror("sigprocmask");
        _exit(1);
    }
}

/* ================== Gestion de interrupcion (SIGINT, solo CHARLIE) ==============
===== */

static volatile sig_atomic_t hay_sigint_pendiente = 0;

static pid_t pid_bosley_raiz = -1;
static pid_t pid_malo_raiz   = -1;

static void terminar_ejecucion_por_sigint(void);
static void matar_todos_los_malos_registrados(void);

/* ================== Salida por pantalla (write) ================== */

static void imprimir_mensaje(const char *fmt, ...)
{
    char    buf[TAM_BUF_MSJ];
```

```
    va_list ap;
    int     n;

    va_start(ap, fmt);
    n = vsnprintf(buf, sizeof(buf), fmt, ap);
    va_end(ap);

    if (n <= 0)
        return;
    if (n > (int)sizeof(buf))
        n = (int)sizeof(buf);

    ssize_t escritos = 0;
    while (escritos < n) {
        ssize_t r = write(STDOUT_FILENO, buf + escritos, (size_t)(n - escritos));
        if (r < 0) {
            if (errno == EINTR)
                continue;
            break;
        }
        if (r == 0)
            break;
        escritos += r;
    }
}

/* ================== Manejadores SIGUSR1/SIGUSR2 ================== */

static void manejador_usr_despertar(int sig)
{
    (void)sig;
}

static void instalar_manejadores_usr(void)
{
    struct sigaction sa;
    memset(&sa, 0, sizeof(sa));
    sa.sa_handler = manejador_usr_despertar;
    sigemptyset(&sa.sa_mask);
    sa.sa_flags = 0;

    sigaction(SIGUSR1, &sa, NULL);
    sigaction(SIGUSR2, &sa, NULL);
}

/* ================== Manejador SIGTERM del malo ================== */

static void manejador_sigterm_malo(int sig)
{
    (void)sig;
    imprimir_mensaje("MALO: AY, me han dado... pulvis sumus, collige, virgo, rosas\n
");
    _exit(0);
}

/* ================== Manejador de SIGINT (solo CHARLIE) ================== */

static void manejador_sigint_charlie(int sig)
{
    (void)sig;
    hay_sigint_pendiente = 1;
}

/* ================== Numeros aleatorios ================== */

static void inicializar_generador_azar(void)
{
    unsigned int semilla =
        (unsigned int)(time(NULL) ^ ((unsigned long)getpid() << 16));
```

```
137:     srand(semilla);
138: }
139:
140:
141: static int obtener_azar_en_rango(int a, int b)
142: {
143:     if (a >= b)
144:         return a;
145:     return a + (int)(rand() / (1.0 + RAND_MAX) * (b - a + 1));
146: }
147:
148: /* ================== Gestion de velocidad ================== */
149:
150: typedef enum {
151:     VEL_NORMAL = 0,
152:     VEL_VELOZ  = 1
153: } t_velocidad;
154:
155: static t_velocidad modo_velocidad = VEL_NORMAL;
156:
157: static void mostrar_ayuda_y_salir(const char *progname)
158: {
159:     imprimir_mensaje("Uso: %s [normal|veloz]\n", progname);
160:     _exit(1);
161: }
162:
163: static void configurar_modo_velocidad(int argc, char *argv[])
164: {
165:     if (argc == 1) {
166:         modo_velocidad = VEL_NORMAL;
167:         return;
168:     }
169:
170:     if (argc == 2) {
171:         if (strcmp(argv[1], "normal") == 0) {
172:             modo_velocidad = VEL_NORMAL;
173:         } else if (strcmp(argv[1], "veloz") == 0) {
174:             modo_velocidad = VEL_VELOZ;
175:         } else {
176:             mostrar_ayuda_y_salir(argv[0]);
177:         }
178:         return;
179:     }
180:
181:     mostrar_ayuda_y_salir(argv[0]);
182: }
183:
184:
185: static void dormir_intervalo(int min_s, int max_s)
186: {
187:     if (modo_velocidad == VEL_VELOZ)
188:         return;
189:
190:     int t = obtener_azar_en_rango(min_s, max_s);
191:     while (t > 0) {
192:         unsigned int r = sleep((unsigned int)t);
193:         if (r == 0)
194:             break;
195:         t = (int)r;
196:     }
197: }
198:
199: /* ================== Fichero pids.bin ================== */
200:
201:
202: static void crear_fichero_tabla_pids(void)
203: {
204:     int fd_pids;
205:     size_t tam_tabla_pids = NUM_SLOTS_PIDS * sizeof(int32_t);
```

```
  206:     int32_t tabla_pids_inicial[NUM_SLOTS_PIDS];
  207:
  208:     memset(tabla_pids_inicial, 0, sizeof(tabla_pids_inicial));
  209:
  210:     fd_pids = open(FICHERO_PIDS, O_WRONLY | O_CREAT | O_TRUNC, 0600);
  211:     if (fd_pids < 0) {
  212:         perror("CHARLIE: open pids.bin");
  213:         _exit(1);
  214:     }
  215:
  216:     ssize_t bytes_escritos = 0;
  217:     const char *puntero_buffer = (const char *)tabla_pids_inicial;
  218:     while ((size_t)bytes_escritos < tam_tabla_pids) {
  219:         ssize_t r = write(fd_pids,
  220:                           puntero_buffer + bytes_escritos,
  221:                           tam_tabla_pids - (size_t)bytes_escritos);
  222:         if (r < 0) {
  223:             if (errno == EINTR)
  224:                 continue;
  225:             perror("CHARLIE: write inicializando pids.bin");
  226:             close(fd_pids);
  227:             _exit(1);
  228:         }
  229:         if (r == 0)
  230:             break;
  231:         bytes_escritos += r;
  232:     }
  233:
  234:     if ((size_t)bytes_escritos < tam_tabla_pids) {
  235:         imprimir_mensaje("CHARLIE: Error, no se han podido escribir todos los bytes
en %s\n",
  236:                          FICHERO_PIDS);
  237:         close(fd_pids);
  238:         _exit(1);
  239:     }
  240:
  241:     close(fd_pids);
  242: }
  243:
  244: /* ================== Limpieza de todos los MALOS vivos ================== */
  245:
  246: static void matar_todos_los_malos_registrados(void)
  247: {
  248:     int fd_pids = open(FICHERO_PIDS, O_RDONLY);
  249:     if (fd_pids < 0) {
  250:         return;
  251:     }
  252:
  253:     size_t tam_tabla_pids = NUM_SLOTS_PIDS * sizeof(int32_t);
  254:     int32_t *tabla_pids_mapeada =
  255:         (int32_t *)mmap(NULL, tam_tabla_pids, PROT_READ, MAP_SHARED, fd_pids, 0);
  256:     if (tabla_pids_mapeada == MAP_FAILED) {
  257:         perror("mmap pids.bin en matar_todos_los_malos");
  258:         close(fd_pids);
  259:         return;
  260:     }
  261:     close(fd_pids);
  262:
  263:     int indice;
  264:     for (indice = 0; indice < NUM_SLOTS_PIDS; ++indice) {
  265:         if (tabla_pids_mapeada[indice] != 0) {
  266:
  267:             kill((pid_t)tabla_pids_mapeada[indice], SIGKILL);
  268:         }
  269:     }
  270:
  271:     munmap(tabla_pids_mapeada, tam_tabla_pids);
  272: }
  273:
```

```
274: /* ================== Prototipos de roles ================== */
275:
276: static int ejecutar_charlie(int argc, char *argv[]);
277: static int ejecutar_bosley(int argc, char *argv[]);
278: static int ejecutar_angel(const char *nombre, int argc, char *argv[]);
279: static int ejecutar_malo(int argc, char *argv[]);
280:
281: /* ================== Helpers de sincronizacion ================== */
282:
283:
284: static void esperar_senal_sincronizacion(int signo)
285: {
286:     sigset_t mascara_espera = mascara_senales_bloqueo;
287:     sigdelset(&mascara_espera, signo);
288:     (void)sigsuspend(&mascara_espera);
289: }
290:
291:
292: static void esperar_senal_charlie_o_ctrlc(int signo)
293: {
294:     for (;;) {
295:         if (hay_sigint_pendiente)
296:             terminar_ejecucion_por_sigint();
297:
298:         sigset_t mascara_espera = mascara_senales_bloqueo;
299:         sigdelset(&mascara_espera, signo);
300:         (void)sigsuspend(&mascara_espera);
301:
302:         if (hay_sigint_pendiente)
303:             terminar_ejecucion_por_sigint();
304:
305:         break;
306:     }
307: }
308:
309: /* ================== CHARLIE: finalizacion ordenada por Ctrl-C ==================
*/
310:
311: static void terminar_ejecucion_por_sigint(void)
312: {
313:     int estado_hijo;
314:
315:
316:     struct sigaction accion_ignorar_sigterm;
317:     memset(&accion_ignorar_sigterm, 0, sizeof(accion_ignorar_sigterm));
318:     accion_ignorar_sigterm.sa_handler = SIG_IGN;
319:     sigemptyset(&accion_ignorar_sigterm.sa_mask);
320:     accion_ignorar_sigterm.sa_flags = 0;
321:     if (sigaction(SIGTERM, &accion_ignorar_sigterm, NULL) < 0) {
322:         perror("CHARLIE: sigaction SIGTERM");
323:         _exit(1);
324:     }
325:
326:
327:     if (kill(0, SIGTERM) < 0) {
328:         perror("CHARLIE: kill(0, SIGTERM)");
329:     }
330:
331:
332:     matar_todos_los_malos_registrados();
333:
334:
335:     while (waitpid(-1, &estado_hijo, 0) > 0)
336:         ;
337:
338:     imprimir_mensaje("Programa interrumpido\n");
339:     _exit(1);
340: }
341:
```

```
  342: /* ================== Codigo de CHARLIE ================== */
  343:
  344: static int ejecutar_charlie(int argc, char *argv[])
  345: {
  346:     pid_t pid_bosley;
  347:     pid_t pid_malo;
  348:     int   status_malo;
  349:     int   status_bosley;
  350:
  351:     configurar_modo_velocidad(argc, argv);
  352:
  353:
  354:     struct sigaction accion_sigint;
  355:     memset(&accion_sigint, 0, sizeof(accion_sigint));
  356:     accion_sigint.sa_handler = manejador_sigint_charlie;
  357:     sigemptyset(&accion_sigint.sa_mask);
  358:     accion_sigint.sa_flags = 0;
  359:     sigaction(SIGINT, &accion_sigint, NULL);
  360:
  361:
  362:
  363:     pid_bosley = fork();
  364:     if (pid_bosley < 0) {
  365:         perror("CHARLIE: fork Bosley");
  366:         _exit(1);
  367:     }
  368:
  369:     if (pid_bosley == 0) {
  370:         char *args_b[3];
  371:
  372:         args_b[0] = (char *)"bosley";
  373:         args_b[1] = (modo_velocidad == VEL_VELOZ) ? (char *)"veloz" : (char *)"norma
l";
  374:         args_b[2] = NULL;
  375:
  376:         if (ruta_ejecutable_programa == NULL) {
  377:             _exit(1);
  378:         }
  379:
  380:         execv(ruta_ejecutable_programa, args_b);
  381:         perror("execv bosley");
  382:         _exit(1);
  383:     }
  384:
  385:     pid_bosley_raiz = pid_bosley;
  386:
  387:     imprimir_mensaje("CHARLIE: Bosley, hijo de mis entretelas, tu PID es %ld. Espero
 a que me avises...\n",
  388:                      (long)pid_bosley);
  389:
  390:
  391:     esperar_senal_charlie_o_ctrlc(SIGUSR1);
  392:
  393:     imprimir_mensaje("CHARLIE: Veo que los Angeles ya han nacido. Creo al malo...\n"
);
  394:
  395:
  396:     crear_fichero_tabla_pids();
  397:
  398:
  399:
  400:     pid_malo = fork();
  401:     if (pid_malo < 0) {
  402:         perror("CHARLIE: fork Malo");
  403:         _exit(1);
  404:     }
  405:
  406:     if (pid_malo == 0) {
  407:         char *args_m[3];
```

```
  408:
  409:         args_m[0] = (char *)"malo";
  410:         args_m[1] = (modo_velocidad == VEL_VELOZ) ? (char *)"veloz" : (char *)"norma
l";
  411:         args_m[2] = NULL;
  412:
  413:         if (ruta_ejecutable_programa == NULL) {
  414:             _exit(1);
  415:         }
  416:
  417:         execv(ruta_ejecutable_programa, args_m);
  418:         perror("execv malo");
  419:         _exit(1);
  420:     }
  421:
  422:     pid_malo_raiz = pid_malo;
  423:
  424:     imprimir_mensaje("CHARLIE: El malo ha nacido y su PID es %ld. Aviso a Bosley\n",
  425:                      (long)pid_malo);
  426:
  427:
  428:     if (kill(pid_bosley, SIGUSR2) < 0) {
  429:         perror("CHARLIE: kill(SIGUSR2 a Bosley)");
  430:     }
  431:
  432:
  433:     for (;;) {
  434:         if (hay_sigint_pendiente) {
  435:             terminar_ejecucion_por_sigint();
  436:         }
  437:
  438:         pid_t pid_terminado = waitpid(pid_malo, &status_malo, 0);
  439:         if (pid_terminado < 0) {
  440:             if (errno == EINTR) {
  441:                 if (hay_sigint_pendiente) {
  442:                     terminar_ejecucion_por_sigint();
  443:                 }
  444:                 continue;
  445:             }
  446:             perror("CHARLIE: waitpid(malo)");
  447:             _exit(1);
  448:         }
  449:         break;
  450:     }
  451:
  452:
  453:     for (;;) {
  454:         if (hay_sigint_pendiente) {
  455:             terminar_ejecucion_por_sigint();
  456:         }
  457:
  458:         pid_t pid_terminado = waitpid(pid_bosley, &status_bosley, 0);
  459:         if (pid_terminado < 0) {
  460:             if (errno == EINTR) {
  461:                 if (hay_sigint_pendiente) {
  462:                     terminar_ejecucion_por_sigint();
  463:                 }
  464:                 continue;
  465:             }
  466:             perror("CHARLIE: waitpid(bosley)");
  467:             _exit(1);
  468:         }
  469:         break;
  470:     }
  471:
  472:     int mascara_resultado = 0;
  473:     if (WIFEXITED(status_bosley)) {
  474:         mascara_resultado = WEXITSTATUS(status_bosley) & 0x7;
  475:     }
```

```
  476:
  477:     /* ========== Mensaje final UNICO de Charlie ========== */
  478:
  479:     switch (mascara_resultado) {
  480:     case 0:
  481:         imprimir_mensaje("CHARLIE: El pAjaro volO. Ahora se pone tibio a daiquiris e
n el Caribe\n");
  482:         break;
  483:
  484:     case RES_SABRINA:
  485:         imprimir_mensaje("CHARLIE: Bien hecho, Sabrina, siempre fuiste mi favorita\n
");
  486:         break;
  487:
  488:     case RES_JILL:
  489:         imprimir_mensaje("CHARLIE: Jill, donde pones el ojo, pones la bala\n");
  490:         break;
  491:
  492:     case RES_KELLY:
  493:         imprimir_mensaje("CHARLIE: Bravo por Kelly\n");
  494:         break;
  495:
  496:     case RES_SABRINA | RES_JILL:
  497:         imprimir_mensaje("CHARLIE: Kelly, mala suerte, tus compaNeras acertaron y tU
, no\n");
  498:         break;
  499:
  500:
  501:     case RES_JILL | RES_KELLY:
  502:         imprimir_mensaje("CHARLIE: Sabrina, otra vez serA, te apuntarE a una academi
a de tiro\n");
  503:         break;
  504:
  505:     case RES_SABRINA | RES_KELLY:
  506:         imprimir_mensaje("CHARLIE: Jill, no te preocupes, las pistolas no suelen ");
  507:         imprimir_mensaje("funcionar cuando mAs lo necesitas\n");
  508:         break;
  509:
  510:     case RES_SABRINA | RES_JILL | RES_KELLY:
  511:         imprimir_mensaje("CHARLIE: Pobre malo. Le habEis dejado como un colador... S
ois unos Angeles letales\n");
  512:         break;
  513:     }
  514:
  515:
  516:     matar_todos_los_malos_registrados();
  517:
  518:     return 0;
  519: }
  520:
  521: /* ================== Codigo de BOSLEY ================== */
  522:
  523: static int ejecutar_bosley(int argc, char *argv[])
  524: {
  525:     static const char *nombres_angeles[3] = { "sabrina", "jill", "kelly" };
  526:     pid_t pids_angeles[3];
  527:     int i;
  528:
  529:     configurar_modo_velocidad(argc, argv);
  530:
  531:     imprimir_mensaje("BOSLEY: Hola, papA, dOnde estA mamA? Mi PID es %ld y voy a cre
ar a los Angeles...\n",
  532:                      (long)getpid());
  533:
  534:
  535:     for (i = 0; i < 3; i++) {
  536:         pid_t pid = fork();
  537:         if (pid < 0) {
  538:             perror("BOSLEY: fork angel");
```

```
  539:             _exit(1);
  540:         }
  541:
  542:         if (pid == 0) {
  543:             char *args_a[3];
  544:
  545:             args_a[0] = (char *)nombres_angeles[i];
  546:             args_a[1] = (modo_velocidad == VEL_VELOZ) ? (char *)"veloz" : (char *)"n
ormal";
  547:             args_a[2] = NULL;
  548:
  549:             if (ruta_ejecutable_programa == NULL) {
  550:                 _exit(1);
  551:             }
  552:
  553:             execv(ruta_ejecutable_programa, args_a);
  554:             perror("execv angel");
  555:             _exit(1);
  556:         } else {
  557:             pids_angeles[i] = pid;
  558:         }
  559:     }
  560:
  561:
  562:     kill(getppid(), SIGUSR1);
  563:
  564:
  565:     esperar_senal_sincronizacion(SIGUSR2);
  566:
  567:
  568:     for (i = 0; i < 3; i++) {
  569:         if (kill(pids_angeles[i], SIGUSR2) < 0) {
  570:             perror("BOSLEY: kill(SIGUSR2 a angel)");
  571:         }
  572:     }
  573:
  574:
  575:     int vivos = 3;
  576:     int resultado = 0;
  577:
  578:     while (vivos > 0) {
  579:         int status;
  580:         pid_t pid = wait(&status);
  581:         if (pid < 0) {
  582:             if (errno == EINTR)
  583:                 continue;
  584:             perror("BOSLEY: wait");
  585:             break;
  586:         }
  587:
  588:         int exito = 0;
  589:         if (WIFEXITED(status) && WEXITSTATUS(status) == 0) {
  590:             exito = 1;
  591:         }
  592:
  593:         if (pid == pids_angeles[0]) {
  594:             if (exito) resultado |= RES_SABRINA;
  595:         } else if (pid == pids_angeles[1]) {
  596:             if (exito) resultado |= RES_JILL;
  597:         } else if (pid == pids_angeles[2]) {
  598:             if (exito) resultado |= RES_KELLY;
  599:         }
  600:
  601:         vivos--;
  602:     }
  603:
  604:     imprimir_mensaje("BOSLEY: Los tres Angeles han acabado su misiOn. Informo del re
sultado a Charlie y muero\n");
  605:
```

```
  606:     _exit(resultado & 0xFF);
  607: }
  608:
  609: /* ================== Codigo del MALO (reencarnaciones + mmap) ==================
*/
  610:
  611: static int ejecutar_malo(int argc, char *argv[])
  612: {
  613:     int fd;
  614:     size_t tam = NUM_SLOTS_PIDS * sizeof(int32_t);
  615:     int32_t *tabla_pids;
  616:
  617:     configurar_modo_velocidad(argc, argv);
  618:
  619:     fd = open(FICHERO_PIDS, O_RDWR);
  620:     if (fd < 0) {
  621:         perror("MALO: open pids.bin");
  622:         _exit(1);
  623:     }
  624:
  625:     tabla_pids = (int32_t *)mmap(NULL, tam,
  626:                                  PROT_READ | PROT_WRITE,
  627:                                  MAP_SHARED, fd, 0);
  628:     if (tabla_pids == MAP_FAILED) {
  629:         perror("MALO: mmap pids.bin");
  630:         close(fd);
  631:         _exit(1);
  632:     }
  633:     close(fd);
  634:
  635:     inicializar_generador_azar();
  636:     int generacion;
  637:
  638:     for (generacion = 1; generacion <= 20; generacion++) {
  639:
  640:         pid_t pid_hijo = fork();
  641:         if (pid_hijo < 0) {
  642:             perror("MALO: fork");
  643:             munmap(tabla_pids, tam);
  644:             _exit(1);
  645:         }
  646:
  647:         if (pid_hijo == 0) {
  648:
  649:
  650:
  651:             struct sigaction sa;
  652:             memset(&sa, 0, sizeof(sa));
  653:             sa.sa_handler = manejador_sigterm_malo;
  654:             sigemptyset(&sa.sa_mask);
  655:             sa.sa_flags = 0;
  656:             if (sigaction(SIGTERM, &sa, NULL) < 0) {
  657:                 perror("MALO: sigaction SIGTERM en hijo");
  658:                 _exit(1);
  659:             }
  660:
  661:             imprimir_mensaje("MALO: JA, JA, JA, me acabo de reencarnar y mi nuevo PI
D es: ");
  662:             imprimir_mensaje("%ld. QuE malo que soy...\n",
  663:                              (long)getpid());
  664:
  665:
  666:             if (generacion >= 1 && generacion <= NUM_SLOTS_PIDS) {
  667:                 int libre = -1;
  668:                 int intentos;
  669:
  670:                 for (intentos = 0; intentos < 40; ++intentos) {
  671:                     int idx = obtener_azar_en_rango(0, NUM_SLOTS_PIDS - 1);
  672:                     if (tabla_pids[idx] == 0) {
```

```
673:                         libre = idx;
674:                         break;
675:                     }
676:                 }
677:                 int idx;
678:
679:                 if (libre < 0) {
680:                     for (idx = 0; idx < NUM_SLOTS_PIDS; ++idx) {
681:                         if (tabla_pids[idx] == 0) {
682:                             libre = idx;
683:                             break;
684:                         }
685:                     }
686:                 }
687:                 if (libre >= 0) {
688:                     tabla_pids[libre] = (int32_t)getpid();
689:                 }
690:             }
691:
692:
693:             dormir_intervalo(1, 3);
694:
695:             _exit(0);
696:         } else {
697:
698:             int status;
699:             for (;;) {
700:                 pid_t r = waitpid(pid_hijo, &status, 0);
701:                 if (r < 0) {
702:                     if (errno == EINTR)
703:                         continue;
704:                     perror("MALO: waitpid(generacion)");
705:                     munmap(tabla_pids, tam);
706:                     _exit(1);
707:                 }
708:                 break;
709:             }
710:         }
711:     }
712:
713:
714:     imprimir_mensaje("MALO: He sobrevivido a mi vigEsima reencarnaciOn. Hago mutis p
or el foro\n");
715:     munmap(tabla_pids, tam);
716:     _exit(0);
717: }
718:
719:
720: /* ================== Angeles ================== */
721:
722: static int ejecutar_angel(const char *nombre, int argc, char *argv[])
723: {
724:     int fd;
725:     size_t tam = NUM_SLOTS_PIDS * sizeof(int32_t);
726:     const int32_t *tabla_pids;
727:     int disparo;
728:     int exito_alguna_vez = 0;
729:
730:     configurar_modo_velocidad(argc, argv);
731:     inicializar_generador_azar();
732:
733:     imprimir_mensaje("%s: Hola, he nacido y mi PID es %ld\n",
734:                      nombre, (long)getpid());
735:
736:
737:     esperar_senal_sincronizacion(SIGUSR2);
738:
739:
740:     fd = open(FICHERO_PIDS, O_RDONLY);
```

```
  741:     if (fd < 0) {
  742:         perror("ANGEL: open pids.bin");
  743:         _exit(1);
  744:     }
  745:
  746:     tabla_pids = (const int32_t *)mmap(NULL, tam,
  747:                                        PROT_READ,
  748:                                        MAP_SHARED, fd, 0);
  749:     if (tabla_pids == MAP_FAILED) {
  750:         perror("ANGEL: mmap pids.bin");
  751:         close(fd);
  752:         _exit(1);
  753:     }
  754:     close(fd);
  755:
  756:     for (disparo = 1; disparo <= 3; disparo++) {
  757:
  758:
  759:         dormir_intervalo(6, 12);
  760:
  761:         int idx    = obtener_azar_en_rango(0, NUM_SLOTS_PIDS - 1);
  762:         int32_t valor = tabla_pids[idx];
  763:
  764:         if (valor == 0) {
  765:             imprimir_mensaje("%s: Pardiez! La pistola se ha encasquillado\n", nombre
);
  766:             continue;
  767:         }
  768:
  769:         pid_t objetivo = (pid_t)valor;
  770:
  771:         imprimir_mensaje("%s: Voy a disparar al PID %ld\n",
  772:                          nombre, (long)objetivo);
  773:
  774:         if (kill(objetivo, SIGTERM) == 0) {
  775:
  776:             imprimir_mensaje("%s: BINGO! He hecho diana! Un malo menos\n", nombre);
  777:             exito_alguna_vez = 1;
  778:
  779:         } else {
  780:             if (errno == ESRCH) {
  781:
  782:                 imprimir_mensaje("%s: He fallado. Vuelvo a intentarlo\n", nombre);
  783:             } else {
  784:
  785:                 perror("ANGEL: kill(SIGTERM al malo)");
  786:                 imprimir_mensaje("%s: He fallado ya tres veces y no me quedan mAs ba
las. Muero\n", nombre);
  787:                 munmap((void *)tabla_pids, tam);
  788:                 _exit(1);
  789:             }
  790:         }
  791:     }
  792:
  793:
  794:     if (exito_alguna_vez) {
  795:
  796:         munmap((void *)tabla_pids, tam);
  797:         _exit(0);
  798:     } else {
  799:         imprimir_mensaje("%s: He fallado ya tres veces y no me quedan mAs balas. Mue
ro\n", nombre);
  800:         munmap((void *)tabla_pids, tam);
  801:         _exit(1);
  802:     }
  803: }
  804:
  805:
  806: /* ================== main general ================== */
```

```
807:
808: int main(int argc, char *argv[])
809: {
810:     bloquear_senales_de_arranque();
811:     instalar_manejadores_usr();
812:
813:     const char *ruta_env_ejecutable = getenv("CHARLIE_PATH");
814:     if (ruta_env_ejecutable != NULL && ruta_env_ejecutable[0] != '\0') {
815:         ruta_ejecutable_programa = ruta_env_ejecutable;
816:     } else {
817:         ruta_ejecutable_programa = argv[0];
818:         (void)setenv("CHARLIE_PATH", ruta_ejecutable_programa, 1);
819:     }
820:
821:     const char *nombre_programa = argv[0];
822:     const char *ultima_barra = strrchr(nombre_programa, '/');
823:     if (ultima_barra != NULL)
824:         nombre_programa = ultima_barra + 1;
825:
826:     if (strcmp(nombre_programa, "charlie") == 0) {
827:         return ejecutar_charlie(argc, argv);
828:     } else if (strcmp(nombre_programa, "bosley") == 0) {
829:         return ejecutar_bosley(argc, argv);
830:     } else if (strcmp(nombre_programa, "sabrina") == 0) {
831:         return ejecutar_angel("SABRINA", argc, argv);
832:     } else if (strcmp(nombre_programa, "jill") == 0) {
833:         return ejecutar_angel("JILL", argc, argv);
834:     } else if (strcmp(nombre_programa, "kelly") == 0) {
835:         return ejecutar_angel("KELLY", argc, argv);
836:     } else if (strcmp(nombre_programa, "malo") == 0) {
837:         return ejecutar_malo(argc, argv);
838:     }
839:
840:     return ejecutar_charlie(argc, argv);
841: }
```