Fundamentos de Sistemas Operativos

tercera práctica

Explotación programada de "sistemas operativos de tipo UNIX" mediante el desarrollo de programas que hacen uso de la funcionalidad proporcionada directamente por el núcleo

(llamadas a sistema)

Escuela Politécnica Superior de Alcoy
Curso 2021 – 2022

Tercera práctica: llamadas a sistema

Objetivos

- Adquirir competencia en la explotación programada de “sistemas operativos de tipo UNIX” mediante el desarrollo de programas que hacen uso de la funcionalidad proporcionada directamente por el núcleo.
- Asimilar el concepto “intérprete de comandos de tipo UNIX” y los principios de explotación UNIX “redirección de entrada/salida” y “combinación de comandos”.

Aspectos ejercitados

- Gestión básica de procesos (las llamadas a sistema `fork`(), `exit`(), `wait`() y `exec`()).
- Gestión básica de ficheros (llamadas `open`(), `close`(), `read`() y `write`()).
- Redirección de entrada salida (llamada `dup2`()).
- Comunicación básica entre procesos (llamada `pipe`()).
- Señalización básica entre procesos (llamadas `kill`(), `alarm`(), `sigprocmask`(), `sigsuspend`() y `sigaction`()).

Contenido

Ejercicios para evaluación

A continuación se propone un conjunto de ejercicios que el alumno podrá realizar y, en su caso, entregar al profesor.

Algunos ejercicios a realizar dependen del número de DNI del alumno. Dicho número consta de 8 dígitos. Anotarlos, como se indica en el ejemplo, y adjuntarlos a la documentación presentada sobre las prácticas.

Ejemplo:

Digito	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**
DNI	4	4	8	6	8	5	2	7

DNI alumno:

Digito	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**
DNI								

Cada vez que, en un ejercicio, se indique *dígito_i* (siendo *i* un número entre 1 y 8), la expresión deberá ser sustituida por el valor consignado en la casilla, del DNI del alumno, correspondiente al dígito i. Así, según los datos mostrados, en el ejemplo de DNI, si en un ejercicio se pide crear *dígito_3* directorios distintos, se deberá crear 5 directorios distintos (pues en el DNI del ejemplo tenemos el valor 5 correspondiendo al dígito 3). Si, al sustituir *dígito_i*, obtenemos 0 tomar 1 en su lugar.

Para optar a la máxima nota, es suficiente con entregar tres de los cinco ejercicios propuestos (a elegir).

Trabajo previo: Crear un fichero de texto con diez líneas de caracteres alfanuméricos, elegidos al azar (alguna decena de caracteres en cada línea y, entre todas las líneas, los diez dígitos, al menos, una vez), y llamarlo *fichero_previo*, en el directorio de trabajo por defecto. Sin cambiar de directorio de trabajo por defecto (ni ahora, ni durante el desarrollo de la práctica), solicitarle, a un “intérprete de comandos de tipo UNIX”, la ejecución del siguiente comando:

```
$ cat fichero_previo
```

Anotar el resultado y adjuntarlo a la documentación presentada sobre las prácticas.

EJERCICIOS

1. Crear un fichero fuente con el código (en lenguaje C) que aparece en esta página (y la siguiente) y llamarlo *misterioso_1.c*, en el directorio de trabajo por defecto. Compilar y enlazar el código *misterioso_1.c* y nombrar al fichero ejecutable resultante, en el directorio de trabajo por defecto, con el nombre *misterioso_1*. Sin cambiar de directorio de trabajo por defecto, solicitarle, a un "intérprete de comandos de tipo UNIX", la ejecución del siguiente comando:

```
$ ./misterioso_1 grep digito_1 fichero_previo
```

a) Anotar, detalladamente, el comportamiento observado.

b) Explicar, al más alto nivel de abstracción posible, y sin omitir ningún aspecto funcional relevante, el comportamiento observado.

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

int main(int argc, char *argv[])
{
   pid_t childpid;
   int status, exit_code, x;

   exit_code = EXIT_SUCCESS;

   if (argc < 2)
   {
      printf("Usage: %s command args\n", argv[0]);
      exit_code = EXIT_FAILURE;
   }
   else
   {
      switch (childpid = fork())
      {
         case -1:
            perror("Could not fork\n");
            exit_code = EXIT_FAILURE;
            break;
         case 0:
            if (execvp(argv[1], &argv[1]) < 0)
            {
               perror("Could not execute the command\n");
               exit_code = EXIT_FAILURE;
               break;
            }
```

```
            default:
               if ((x = wait(&status)) != childpid)
               {
                  perror("Wait interrupted by a signal\n");
                  exit_code = EXIT_FAILURE;
               }
         } // end switch
      } // end else
      exit(exit_code);
   } // end main
```

2. Crear un fichero fuente con el código (en lenguaje C) que aparece en esta página (y la siguiente) y llamarlo *misterioso_2.c*, en el directorio de trabajo por defecto. Compilar y enlazar el código *misterioso_2.c* y nombrar al fichero ejecutable resultante, en el directorio de trabajo por defecto, con el nombre *misterioso_2*. Sin cambiar de directorio de trabajo por defecto, y en una situación en la que, en el directorio de trabajo por defecto, no existe una entrada con el nombre fichero_2, solicitarle, a un "intérprete de comandos de tipo UNIX", la ejecución de los siguientes comandos:

```
$ ./misterioso_2 fichero_previo fichero_2
$ tail –n digito_2 fichero_2
```

a) Anotar, detalladamente, el comportamiento observado.
b) Explicar, al más alto nivel de abstracción posible, y sin omitir ningún aspecto funcional relevante, el comportamiento observado.

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

#define TRUE 1
#define FALSE 0
#define BLKSIZE 1
#define NEWFILE (O_WRONLY | O_CREAT | O_EXCL)
#define MODE600 (S_IRUSR | S_IWUSR)

int main(int argc, char *argv[])
{
   int from_fd, to_fd, count, exit_code;
   int written_flag, read_flag, end_of_file_flag;

   char buf[BLKSIZE];

   exit_code = EXIT_SUCCESS;

   if (argc != 3)
   {
      printf("Usage: %s from_file to_file\n", argv[0]);
      exit_code = EXIT_FAILURE;
   }
   else
   {
      if ((from_fd = open(argv[1], O_RDONLY)) < 0)
      {
         perror("Could not open the source file\n");
         exit_code = EXIT_FAILURE;
      }
      else
      {
         if ((to_fd = open(argv[2], NEWFILE, MODE600)) < 0)
         {
            perror("Could not create the destination file\n");
            exit_code = EXIT_FAILURE;
         }
```

```
            else
            {
               written_flag = TRUE;
               read_flag = TRUE;
               end_of_file_flag = FALSE;
               while (written_flag && read_flag && !end_of_file_flag)
               {
                  switch (count = read(from_fd, buf, sizeof(buf)))
                  {
                     case -1:
                        read_flag = FALSE;
                        perror("Could not read from source\n");
                        exit_code = EXIT_FAILURE;
                        break;
                     case 0:
                        end_of_file_flag = TRUE;
                        break;
                     default:
                        if (write(to_fd, buf, count) != count)
                        {
                           written_flag = FALSE;
                           perror("Could not write to destination\n");
                           exit_code = EXIT_FAILURE;
                        }
                  } //switch
               } // while
               if (close(to_fd) != 0)
               {
                  perror("Could not close the destination file\n");
                  exit_code = EXIT_FAILURE;
               }
            } // else open to_fd
            if (close(from_fd) != 0)
            {
               perror("Could not close the source file\n");
               exit_code = EXIT_FAILURE;
            }
         } // else open from_fd
      } // else argc
      exit(exit_code);
   } // main
```

3. Crear un fichero fuente con el código (en lenguaje C) que aparece en esta página (y la siguiente) y llamarlo *misterioso_3.c*, en el directorio de trabajo por defecto. Compilar y enlazar el código *misterioso_3.c* y nombrar al fichero ejecutable resultante, en el directorio de trabajo por defecto, con el nombre *misterioso_3*. Sin cambiar de directorio de trabajo por defecto, y en una situación en la que, en el directorio de trabajo por defecto, no existe una entrada con el nombre fichero_3, solicitarle, a un "intérprete de comandos de tipo UNIX", la ejecución de los siguientes comandos:

```
$ ./misterioso_3 fichero_3 tail –n digito_3 fichero_previo
$ head –n digito_2 fichero_3
```

a) Anotar, detalladamente, el comportamiento observado.
b) Explicar, al más alto nivel de abstracción posible, y sin omitir ningún aspecto funcional relevante, el comportamiento observado.

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

#define NEWFILE (O_WRONLY | O_CREAT | O_EXCL)
#define MODE644 (S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH)

int redirect_output(const char *file)
{
   int return_code, fd;

   return_code = 0;

   if ((fd = open(file, NEWFILE, MODE644)) < 0)
   {
      return_code = -1;
   }
   else
   {
      if (dup2(fd, STDOUT_FILENO) < 0)
      {
         return_code = -1;
      }
      else
      {
         if (close(fd) != 0)
         {
            return_code = -1;
         }
      }
   }
   return return_code;
}
```

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

   exit_code = EXIT_SUCCESS;

   if (argc < 3)
   {
      printf("Usage: %s to_file command args\n", argv[0]);
      exit_code = EXIT_FAILURE;
   }
   else
   {
      if (redirect_output(argv[1]) == -1)
      {
         perror("Could not redirect the output\n");
         exit_code = EXIT_FAILURE;
      }
      else
      {
         if (execvp(argv[2], &argv[2]) < 0)
         {
            perror("Could not execute the command\n");
            exit_code = EXIT_FAILURE;
         }
      }
   }
   exit(exit_code);
}
```

4. Crear un fichero fuente con el código (en lenguaje C) que aparece en esta página (y la siguiente) y llamarlo *misterioso_4.c*, en el directorio de trabajo por defecto. Compilar y enlazar el código *misterioso_4.c* y nombrar al fichero ejecutable resultante, en el directorio de trabajo por defecto, con el nombre *misterioso_4*. Sin cambiar de directorio de trabajo por defecto, solicitarle, a un "intérprete de comandos de tipo UNIX", la ejecución del siguiente comando:

```
$ ./misterioso_4 dígito_2
```

a) Anotar, detalladamente, el comportamiento observado.

b) Explicar, al más alto nivel de abstracción posible, y sin omitir ningún aspecto funcional relevante, el comportamiento observado.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <errno.h>
#include <string.h>
#include <sys/types.h>
#include <sys/wait.h>
#define BUFSIZE 256

int main(int argc, char *argv[])
{
   pid_t childpid;
   int fd[2], status, exit_code, a, b, x;
   char buf[BUFSIZE];
   unsigned strsize;

   exit_code = EXIT_SUCCESS;

   if (argc != 2)
   {
      printf("Usage: %s digito_2\n", argv[0]);
      exit_code = EXIT_FAILURE;
   }
   else
   {
      if (pipe(fd) != 0)
      {
         perror("Could not create the pipe\n");
         exit_code = EXIT_FAILURE;
      }
      else
      {
         switch (childpid = fork())
         {
            case -1:
               perror("Could not fork\n");
               exit_code = EXIT_FAILURE;
               break;
            case 0:
               if (dup2(fd[1], STDOUT_FILENO) < 0)
               {
                  perror("Child: fd duplication failed\n");
                  exit_code = EXIT_FAILURE;
               }
               else
```

```
                {
                   close(fd[0]); // no deberia de producirse error
                   close(fd[1]); // no deberia de producirse error
                   a = (int) getpid();
                   b = atoi(argv[1]);
                   sprintf(buf, "[%d],[%d]\n", a, a % b);
                   strsize = strlen(buf) + 1;
                   if (write(STDOUT_FILENO, buf, strsize) != strsize)
                   {
                      perror("Child: write to pipe failed\n");
                      exit_code = EXIT_FAILURE;
                   }
                } // else
                break;
             default:
                if (dup2(fd[0], STDIN_FILENO) < 0)
                {
                   perror("Parent: fd duplication failed\n");
                   exit_code = EXIT_FAILURE;
                }
                else
                {
                   close(fd[0]); // no deberia de producirse error
                   close(fd[1]); // no deberia de producirse error
                   if (x = wait(&status) != childpid)
                   {
                      perror("Parent: unexpected error\n");
                   }
                   else
                   {
                      if (status == 0)
                      {
                         if (read(STDIN_FILENO, buf, BUFSIZE) <= 0)
                         {
                            perror("Parent: pipe read failed\n");
                            exit_code = EXIT_FAILURE;
                         }
                         else
                         {
                            printf("[%d] got: %s\n", (int) getpid(), buf);
                         }
                      }
                      else
                      {
                         perror("Parent: child had problems\n");
                         exit_code = EXIT_FAILURE;
                      }
                   }
                } // end if default
          } // end switch
       } // end if pipe
    } //end if argc
    exit(exit_code);
} // end main
```

5. Crear un fichero fuente con el código (en lenguaje C) que aparece en esta página (y la siguiente) y llamarlo *misterioso_5.c*, en el directorio de trabajo por defecto. Compilar y enlazar el código *misterioso_5.c* y nombrar al fichero ejecutable resultante, en el directorio de trabajo por defecto, con el nombre *misterioso_5*. Sin cambiar de directorio de trabajo por defecto, solicitarle, a un "intérprete de comandos de tipo UNIX", la ejecución del comando:

```
$ ./misterioso_5 10 dígito_5 100&
```

Anotar el número de proceso en el que se ejecuta el comando solicitado, al cual nos referiremos como *pid*. A lo largo de los siguientes 100 segundos, pedirle, al mismo intérprete de comandos, *dígito_3* veces, la ejecución del comando:

```
$ kill -10 pid
```

a) Anotar, detalladamente, el comportamiento observado.
b) Explicar, al más alto nivel de abstracción posible, y sin omitir ningún aspecto funcional relevante, el comportamiento observado.

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

#define TRUE 1
#define FALSE 0

int timeout_expired, a, b, c;

void handler(int sig_num)
{
   c = c + 1;

   if ( sig_num != SIGALRM )
   {
      printf("[%d] signal catched [%d],[%d]\n", a, c, a%b);
   }
   else
   {
      timeout_expired = TRUE;
      printf("[%d] timeout expired [%d]\n", a, c);
   }
}
```

```
int main(int argc, char *argv[])
{
    struct sigaction act;

    sigset_t sigset;

    int signal_num, num_of_seconds, exit_code;

    if ( argc != 4 )
    {
       printf("Usage: %s sig_num digito_5 num_seconds\n", argv[0]);
       exit_code = EXIT_FAILURE;
    }
    else
    {
       signal_num = atoi(argv[1]);
       b = atoi(argv[2]);
       num_of_seconds = atoi(argv[3]);
       a = (int) getpid();
       c = 0;

       act.sa_handler = handler;
       sigfillset(&act.sa_mask);
       act.sa_flags = 0;
       sigaction(SIGALRM, &act, NULL);
       sigaction(signal_num, &act, NULL);

       sigfillset(&sigset);
       sigdelset(&sigset, signa./misteriosol_num);
       sigdelset(&sigset, SIGALRM);
       printf("waiting for signal n: %s\n", argv[1]);
       alarm(num_of_seconds);
       timeout_expired = FALSE;
       while (timeout_expired == FALSE)
       {
          sigsuspend(&sigset);
       }
       printf("program terminated\n");
       exit_code = EXIT_SUCCESS;
    }
    exit (exit_code);
}
```