Memoria Práctica 2 – Sistemas Operativos

Manuel Suárez Román y Manuel Cintado

Doble Grado Matemáticas e Ingeniería Informática

Contenido

Ejercicio 2

En este ejercicio, tras codificarlo, vemos que cada hijo únicamente es capaz de imprimir el mensaje en el que muestra su identificador, y no la línea de "… y ya me toca terminar" pues ninguno de ellos espera los 30 segundos necesarios, ya que tras 5 segundos de que cada hijo haya sido creado, estos reciben una señal SIGTERM que envía el padre y hace que finalicen sin que alcancen esa parte del código.

Código

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

#define N 4

int main(){
int i;
pid_t pid;
for(i = 0; i < N; ++i){
 pid = fork();
 if(pid < 0){
  printf("Error al mplear e fork\n" );
  exit(EXIT_FAILURE);
 }
 else if( pid == 0){
  printf("Soy el proceso hijo %ld\n", (long) getpid());
  sleep(30);
  printf("Soy el proceso hijo %ld y me toca terminar.\n", (long) getpid());
  exit(EXIT_SUCCESS);
 }
 else{
```

```
   sleep(5);
   kill(pid, SIGTERM);
  }
}
exit(EXIT_SUCCESS);
}
```

Ejercicio 3

a) ¿La llamada a *sigaction* supone que se ejecute la función *manejador*?

No, puesto que la función manejador se ejecuta si el proceso que lo tiene asociado recibe una señal del mismo tipo que se ha especificado como primer parámetro de la función de *sigaction.*

b) ¿Cuándo aparece el *printf* en pantalla?

Puesto que en esta pregunta no queda claro a que *printf* se refiere, si el del main o el del manejador:

Si nos referimos al caso del *printf* del main, cada 9999 aparece por pantalla. Otra opción es recibir una señal SIGINT, tras la cual aparecerá también

Refiriéndonos al *printf* de dentro del manejador, únicamente aparecerá cuando la señal SIGINT sea recibida, es decir, cuando el usuario pulse Ctrl + C.

c) ¿Qué ocurre por defecto cuando un programa recibe una señal y no la tiene capturada?

No está definido un comportamiento, sino 5 posibilidades en función de la señal que hemos recibido, tal y como se nos ha explicado, estas posibles acciones son abortar el proceso, pararlo, continuarlo, finalizarlo o ignorar la señal.

d) ¿Por qué nunca sale por pantalla *"He conseguido capturar SIGKILL"*?

La señal de SIGKILL es de un tipo especial, pues no se puede capturar por las restricciones por seguridad del Sistema Operativo, por lo tanto, al estar este *printf* en el manejador del mismo, no entra nunca en este y nunca se imprime

Código

```
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <sys/types.h>
#include <unistd.h>
/* manejador: rutina de tratamiento de la señal SIGINT. */
void manejador(int sig){
 printf("He conseguido capturar SIGKILL");
 fflush(stdout);
}

int main(void){
 struct sigaction act;
 act.sa_handler = manejador;
 sigemptyset(&(act.sa_mask));
 act.sa_flags =0;
 if(sigaction(SIGKILL,&act,NULL)<0){
  perror("sigaction");
  exit(EXIT_FAILURE);
 }
 while(1){
  printf("En espera de SigKill (PID = %d)\n", getpid());
  sleep(9999);
 }
}
```

Ejercicio 4

Código

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

#define N_PROC 5
/**
 * Manejador de la funcion SIGUSR1, definida para los participantes de la carrera
 */
void manejador_SIGUSR1(int sig){
  printf("PID: %ld ha recibido SIGUSR1\n", (long)getpid());
  exit(EXIT_SUCCESS);
}

/**
 * Manejador de la señar SIGUSR1 definida unicamente para el proceso gestor
 */
void manejador_SIGUSR1b(int sig){
  printf("PID(Gestor): %ld ha recibido SIGUSR1\n", (long)getpid());
  while(wait(NULL)>= 0);
  exit(EXIT_SUCCESS);
}

/**
 * Manejador de la señal USR2 definida unicamente para el proceso gestor y los
 * participantes de la carrera.
 */
void manejador_USR2b(int sig){
    printf("Señal SIGUSR2 recibida en proceso con pid = %ld\n", (long)getpid());
```

```
    fflush(stdout);
}

/**
 * Manejador de la señal USR2 defnida unicamente para el proceso padre.
 */
void manejador_SIGUSR2(int sig){
  printf("Aviso de que la competicion va a comenzar\n");
  if(kill(0, SIGUSR1) < 0){
      perror("kill");
      exit(EXIT_FAILURE);
    }
}

int main(void){
  struct sigaction act;
  pid_t pid;
  int i;

  pid = fork();
  if(pid < 0){
    perror("fork");
    exit(EXIT_FAILURE);
  }
  else if(pid == 0){
    sigemptyset(&(act.sa_mask));
    act.sa_flags = 0;

    act.sa_handler = manejador_SIGUSR1b;
    if(sigaction(SIGUSR1, &act, NULL) < 0){
      perror("sigaction");
```

```
  exit(EXIT_FAILURE);
}

act.sa_handler = manejador_USR2b;
if (sigaction(SIGUSR2, &act, NULL) < 0) { //Establecemos la captura de la señal SIGUSR2
  perror("sigaction");
  exit(EXIT_FAILURE);
}

for(i = 0; i < N_PROC; ++i){
  pid = fork();
  if(pid < 0){
    perror("fork");
    exit(EXIT_FAILURE);
  }
  else if(pid == 0){
      sigemptyset(&(act.sa_mask));
      act.sa_flags = 0;
      act.sa_handler = manejador_SIGUSR1;

      if(sigaction(SIGUSR1, &act, NULL) < 0){
        perror("sigaction");
        exit(EXIT_FAILURE);
      }
      printf("Soy el proceso %ld, estoy listo\n", (long)getpid());

      kill(getppid(), SIGUSR2);
      pause(); //Esperamos a recibir la señal de que la competicion ha comenzado
    }
 else {
   pause(); //Esperamos a que el hijo que acabamos de crear esté listo
```

```
    }
   }
    printf("Aviso desde el proceso gestor de que todos los participantes están listos\n");
    if(kill(getppid(), SIGUSR2) < 0){
     perror("kill");
     exit(EXIT_FAILURE);
    }
    pause(); //Esperamos a recibir la señal de que la competicion ha comenzado
   }
  else{
  /*prcoeso padre tocho*/
   sigemptyset(&(act.sa_mask));
   act.sa_flags = 0;
   act.sa_handler = manejador_SIGUSR2;
   if(sigaction(SIGUSR2, &act, NULL) < 0){
    perror("sigaction");
    exit(EXIT_FAILURE);
   }
   act.sa_handler = SIG_IGN; //Ignoramos la señal USR1
   if (sigaction(SIGUSR1, &act, NULL) < 0) {
    perror("sigaction");
    exit(EXIT_FAILURE);
   }
   pause(); //Esperamos a recibir la señal del proceso gestor
   wait(NULL); //Esperamos a que el hijo gestor termine
   return EXIT_SUCCESS;
  }
 }
```

Ejercicio 6

a) ¿Qué sucede cuando el hijo recibe la señal de alarma?

En el momento en el que la señal SIGALRM es recibida por el hijo, este finaliza a pesar de que esté en medio de un *sleep*, pues esta es una de las características de las Alarmas. Tras esto, el padre finaliza.

Código

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

#define N_ITER 5



int main(void){
 pid_t pid;
 sigset_t set1, set2, setaux;
 int counter;
 pid = fork();



 if(pid <0){
  perror("fork");
  exit(EXIT_FAILURE);
 }

 if(pid ==0){
  alarm(40);
```

```
  /*Entiendo que justo antes de comenzar cada bloque es lo mimso que al pinicpio de cada
bloque*/
   while(1){
    printf("Bloqueando SIGUSR1\n");
    sigaddset(&set1, SIGUSR1);
    printf("Bloqueando SIGUSR2\n");
    sigaddset(&set1, SIGUSR2);
    printf("Bloqueando SIGALARM\n");
    sigaddset(&set1, SIGALRM);

    if (sigprocmask(SIG_BLOCK, &set1, &setaux) < 0) {
       perror("sigprocmask");
       exit(EXIT_FAILURE);
    }
    for(counter = 0;counter < N_ITER;counter++){
     printf("%d\n", counter);
     sleep(1);
    }

    sigaddset(&set2, SIGUSR1);
    sigaddset(&set2, SIGALRM);

    if (sigprocmask(SIG_UNBLOCK, &set2, &setaux) < 0) {
       perror("sigprocmask");
       exit(EXIT_FAILURE);
    }
    sleep(3);
   }
  }
 while(wait(NULL)>0);
```

```
}
```

b)

Código

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

#define N_ITER 5

void manejador_SIGTERM(int sig){
  printf("Soy %ld y he recibido la señal SIGTERM\n", (long)getpid());
  exit(EXIT_SUCCESS);
  return;
}

int main (void){
  pid_t pid;
  int counter;
  struct sigaction act;


  pid = fork();
  if(pid < 0){
    perror("fork");
    exit(EXIT_FAILURE);
  }
  if(pid == 0){
```

```
    while(1){
      act.sa_handler = manejador_SIGTERM;
      act.sa_flags = 0;
      sigemptyset(&(act.sa_mask));
      if(sigaction(SIGTERM, &act, NULL) < 0){
        perror("Sigaction");
        exit(EXIT_FAILURE);
      }
      for(counter = 0; counter < N_ITER; counter++){
        printf("%d\n", counter);
        sleep(1);
      }
      sleep(3);
    }
  }
  else{
    sleep(40);
    kill(pid,SIGTERM);
  }
  while(wait(NULL)>0);
  exit(EXIT_SUCCESS);
}

//Cuando el hijo haya finalizado su ejecucion el apdre terminara
```

Ejercicio 8

a) SECS = 0, N_READ = 1

Con este valor de los parámetros, únicamente se producen escrituras de manera ordenada, es decir, que no se inicia una escritura sin que la anterior haya acabado. Sin embargo, comparándolo con los resultados de mis compañeros hemos visto que este resultado no debería necesariamente ser así. Creo que esto puede deberse a la distribución de Ubuntu en Windows que utilizo en vez de Linux puro como hacen mis compañeros

b) SECS = 1, N_READ = 10

Con este valor de los parámetros, se producen muchas lecturas y, una vez finalizadas todas estas, se realiza una única escritura. No se escribirá hasta que todas las lecturas hayan finalizado correctamente, pues, en caso contrario, algunos procesos hijos estría leyendo información que se estaría añadiendo nuevamente y eso sería incorrecto

c) SECS = 0, N_READ = 10

Con este valor de los parámetros, no se realiza ninguna escritura apenas, únicamente inicios y finalizaciones consecutivas de lecturas, pues al no esperar los hijos y ser 10, probabilísticamente es más fácil que sea un hijo el que ejecuta el código por delante del padre y, por lo tanto, eleve el valor del semáforo y el padre ya no pueda escribir

Código

```
#include <stdio.h>

#include <stdlib.h>

#include <semaphore.h>

#include <fcntl.h>

#include <sys/stat.h>

#include <sys/wait.h>

#include <unistd.h>


#define N_READ 10

#define SECS 0


#define SEM_LECTURA "/sem_lectura"

#define SEM_ESCRITURA "/sem_escritura"

#define LECTORES "/lectores"


void leer(){
```

```
  printf("R-INI %ld\n", (long)getpid());
  sleep(1);
  printf("R-FIN %ld\n", (long)getpid());
}

void escribir(){
  printf("W-INI %ld\n", (long)getpid());
  sleep(1);
  printf("W-FIN %ld\n", (long)getpid());
}


void manejador_SIGINT(int sig){
  /*Mandamos la señal de SIGTERM a todos los hijos*/

  sem_unlink(SEM_LECTURA);
  sem_unlink(SEM_ESCRITURA);
  sem_unlink(LECTORES);

  if(kill(0, SIGTERM) < 0){
    perror("kill");
    exit(EXIT_FAILURE);
  };

  while(wait(NULL)>0);

  exit(EXIT_SUCCESS);
}

void manejador_SIGTERM(int sig){
  printf("Hijo %ld finalizado\n", (long)getpid());
  exit(EXIT_SUCCESS);
  return;
```

```
}

int main(void){
  sem_t *sem_lectura = NULL, *sem_escritura = NULL, *lectores = NULL;
  pid_t pid;
  int i, aux;
  sigset_t set1, setaux;
  struct sigaction act;



  sigemptyset(&(act.sa_mask));
  act.sa_handler = manejador_SIGTERM;
  act.sa_flags = 0;
  if(sigaction(SIGTERM, &act, NULL) < 0){
    perror("Sigaction");
    exit(EXIT_FAILURE);
  }

  if((sem_lectura = sem_open(SEM_LECTURA, O_CREAT | O_EXCL, S_IRUSR | S_IWUSR, 1)) ==
SEM_FAILED){
    perror("sem_open");
    exit(EXIT_FAILURE);
  }
  if((sem_escritura = sem_open(SEM_ESCRITURA, O_CREAT | O_EXCL, S_IRUSR | S_IWUSR, 1)) ==
SEM_FAILED){
    perror("sem_open");
    exit(EXIT_FAILURE);
  }
  if((lectores = sem_open(LECTORES, O_CREAT | O_EXCL, S_IRUSR | S_IWUSR, 0)) == SEM_FAILED){
    perror("sem_open");
    exit(EXIT_FAILURE);
  }

  /*Supongo sin problema que minimo creara un hijo, si no el programa no tiene sentido*/
```

```
pid = fork();
for(i = 1; i < N_READ && pid > 0; ++i){
  pid = fork();
}

if(pid < 0){
  perror("fork");
  exit(EXIT_FAILURE);
}
if(pid == 0){
  act.sa_handler = SIG_IGN;
  if (sigaction(SIGINT, &act, NULL) < 0) {
    perror("sigaction");
    exit(EXIT_FAILURE);
  }
  sigaddset(&set1, SIGTERM);

  while(1){
    if(sem_wait(sem_lectura) == -1){
      perror("sem_wait");
      exit(EXIT_FAILURE);
    }
    if(sem_post(lectores) == -1){
      perror("sem_post");
      exit(EXIT_FAILURE);
    }
    if(sem_getvalue(lectores, &aux) == 0 && aux == 1)
      if(sem_wait(sem_escritura) == -1){
        perror("sem_wait");
        exit(EXIT_FAILURE);
      }
    if(sem_post(sem_lectura) == -1){
      perror("sem_post");
```

```
  exit(EXIT_FAILURE);
}

if (sigprocmask(SIG_BLOCK, &set1, &setaux) < 0) { //Bloqueamos máscara.
  perror("sigprocmask");
  exit(EXIT_FAILURE);
}

leer();

if (sigprocmask(SIG_UNBLOCK, &set1, &setaux) < 0) { //Desbloqueamos máscara.
  perror("sigprocmask");
  exit(EXIT_FAILURE);
}

if(sem_wait(sem_lectura) == -1){
  perror("sem_wait");
  exit(EXIT_FAILURE);
}
if(sem_wait(lectores) == -1){
  perror("sem_wait");
  exit(EXIT_FAILURE);
}
if(sem_getvalue(lectores, &aux) == 0 && aux == 0)
  if(sem_post(sem_escritura) == -1){
    perror("sem_post");
    exit(EXIT_FAILURE);
  }
if(sem_post(sem_lectura) == -1){
  perror("sem_post");
  exit(EXIT_FAILURE);
}
```

```
    sleep(SECS);
  }
  exit(EXIT_SUCCESS);
}
else{
  /*Establezco la señal de interrupcion(SIGINT) y su comportamiento*/
  act.sa_handler = manejador_SIGINT;
  if(sigaction(SIGINT, &act, NULL) < 0){
    perror("Sigaction");
    exit(EXIT_FAILURE);
  }

  act.sa_handler = SIG_IGN;
  if (sigaction(SIGTERM, &act, NULL) < 0) {
    perror("sigaction");
    exit(EXIT_FAILURE);
  }
  sigaddset(&set1, SIGINT);

  while(1){
    if(sem_wait(sem_escritura) == -1){
      perror("sem_wait");
      exit(EXIT_FAILURE);
    }
    if (sigprocmask(SIG_BLOCK, &set1, &setaux) < 0) { //Bloqueamos máscara.
      perror("sigprocmask");
      exit(EXIT_FAILURE);
    }
    escribir();

    if (sigprocmask(SIG_UNBLOCK, &set1, &setaux) < 0) { //Desbloqueamos máscara.
      perror("sigprocmask");
      exit(EXIT_FAILURE);
```

```
      }

      if(sem_post(sem_escritura) == -1){
        perror("sem_post");
        exit(EXIT_FAILURE);
      }

      sleep(SECS);
    }
    sem_close(sem_lectura);
    sem_unlink(SEM_LECTURA);
    sem_close(sem_escritura);
    sem_unlink(SEM_ESCRITURA);
    sem_close(lectores);
    sem_unlink(LECTORES);
    exit(EXIT_SUCCESS);
  }
}
```

Ejercicio 9

Código

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

#define SEM "/semafo"
#define N_PROC 5

sem_t *sem = NULL;

void manejador_SIGTERM(int sig) {
  sem_close(sem);
  exit(EXIT_SUCCESS);
}

/*Manejador para que no haya que borrar manualmente el semaforo de dev/shm cuando hacermos Ctrl+C*/
void manejador_SIGINT(int sig) {
  sem_close(sem);
  sem_unlink(SEM);
  exit(EXIT_SUCCESS);
}

int main(){
  pid_t pid;
  int aux, aux2, i, fin, num_lect[N_PROC];
  FILE *arch;
```

```
sigset_t set1, setaux;
struct sigaction act;

sem_unlink(SEM);
if ((sem = sem_open(SEM, O_CREAT, 0)) == SEM_FAILED) {
   perror("sem_open");
   exit(EXIT_FAILURE);
}
sigemptyset(&(act.sa_mask));
act.sa_flags = 0;
act.sa_handler = manejador_SIGTERM;

if (sigaction(SIGTERM, &act, NULL) < 0) {
  perror("sigaction");
  exit(EXIT_FAILURE);
}
act.sa_handler = manejador_SIGINT;

if (sigaction(SIGINT, &act, NULL) < 0) {
  perror("sigaction");
  exit(EXIT_FAILURE);
}

sigaddset(&set1, SIGTERM);

for(i = 0;i< N_PROC; ++i){
  pid = fork();
  if(pid < 0){
    perror("fork");
    exit(EXIT_FAILURE);
  }
  else if(pid == 0){
    while(1){
      /*De esta manera controlamos que solo un proceso escriba a la vez*/
      if(sem_getvalue(sem, &aux) == 0 && aux == 0){
```

```
        sem_post(sem);
        if (sigprocmask(SIG_BLOCK, &set1, &setaux) < 0) {
          perror("sigprocmask");
          exit(EXIT_FAILURE);
        }

        arch = fopen("file.txt", "a");
        fprintf(arch, "%d\n", i);
        fclose(arch);
        sem_wait(sem);

        if (sigprocmask(SIG_UNBLOCK, &set1, &setaux) < 0) {
           perror("sigprocmask");
           exit(EXIT_FAILURE);
        }
        srand(time(NULL)*getpid());
        /*El 100000 + se puede quitar, pero es para que haga mas de un bucle(para comprobar que todo
fucniona bien)*/
        usleep(100000 + rand()%100000);
      }
    }
  }
 }

 if (sigprocmask(SIG_BLOCK, &set1, &setaux) < 0){
  perror("sigprocmask");
  exit(EXIT_FAILURE);
 }

 sleep(1);
 for(i = 0; i < N_PROC; ++i)
  num_lect[i] = 0;

 fin = -1;
 aux2 = 0;
```

```
while(1){
 /*cuando el padre esta comprobando, ninguno puede escribir*/
 sem_post(sem);
 arch = fopen("file.txt", "r");
 /*El primer que analicemos que lleva mas de 20 gana*/
 while( aux2 != fin && fscanf(arch, "%d", &aux2) != EOF ){
  num_lect[aux2]++;
  if(num_lect[aux2] > 20){
    fin = aux2;
  }
 }

 for(i=0;i<N_PROC;i++)
  printf("Proceso %d, escrituras = %d\n", i, num_lect[i]);

 if(fin != -1){
  printf("Proceso ganador: %d con %d escrituras\n", aux2, num_lect[aux2]);

  kill(0, SIGTERM);

  while(wait(NULL) > 0);

  fclose(arch);
  arch = fopen("file.txt", "w");
  fclose(arch);
  sem_close(sem);
  sem_unlink(SEM);
  return EXIT_SUCCESS;
 }

 /*Reiniciamos el archivo*/
 fclose(arch);
 arch = fopen("file.txt", "w");
 fclose(arch);
```

```
    sem_wait(sem);
    sleep(1);
  }
  return EXIT_SUCCESS;
}
```