PRÁCTICA 2 SOPER

ÁLVARO RODRÍGUEZ RODRIGO
PABLO SANCHEZ FERNANDEZ

1.

El comando kill -L lista las señales que puede mandar kill.
SIGKILL = 9
SIGSTOP = 19

2.

a)

La continuación del código sería
kill(pid,sig);

b)

La terminal dejará de funcionar si se ejecuta SIGSTOP
Y una vez se ejecuta SIGCONT se introducirá lo que hay en stdin en la terminal y te volverá a permitir escribir

3.

a) No, la llamada a sigaction indica la señal que se tiene que enviar para que se llame a la función especificada por `act.sa_handler = handler;`
b) Como tal no se especifica que se bloquee ninguna señal pero por defecto el manejador bloquea la señal que se está llamando. Es decir si se manda SIGINT se bloquea SIGINT durante el manejador
c) Al enviar la señal SIGINT el programa sale del sleep y escribe por pantalla.
d) Como no se pone un manejador al pulsar ctr+c el programa utiliza su manejador por defecto que en este caso es cerrar el programa.
e) struct sigaction act[31];
for(int i = 0;i<31;i++){
act[i].sa_handler = handler
sigemptyset(&(act [i].sa_mask));
act.sa_flags = 0;
if (sigaction (i+1 , & act[i] , NULL) < 0) {
printf(“no pude bloquear la señal: %i”,i+1);
exit(EXIT_FAILURE);
}
No, hay dos señales que no se pueden bloquear ni manejar: SIGKILL y SIGSTOP. Estas dos señales no se pueden manejar dado que debe haber una forma de terminar con la ejecución de un programa.

4.

a) La gestión de la señal se realiza en las primeras lineas de main, en estas lineas se le dice a la estructura act el manejador que va a utilizar, limpia la mascara de señales bloqueadas y pone las flags a 0. Despues se le dice que la señal que va a manejar es SIGINT.

Es en el sleep(9999) donde se espera a que llegue una señal aunque si llegara una señal antes de llegar al sleep(9999) siempre y cuando la línea 21 ya se haya ejecutado, está cambiará el flag got_signal y entrará en el if. Aunque esto solo se hace 1 vez por lo cual se podrían estar perdiendo señales.

b) Se permiten variables globales principalmente porqué es un único proceso que va a tener la información accesible, si hubiera varios procesos al modificar la variable global de uno no se cambiaría en la memoria del otro y no funcionaría

5.

a) Con sigusr1 y sigusr2 no pasa nada por que la máscara están bloqueando las señales. Cuando recibe sigint como no está bloqueada simplemente realiza la señal que hace sigint un exit.
b) cambiar pause() por sleep(10)
   Despues hay que añadir un sigprocmask(SIG_SETMASK,&oset,NULL); de esta forma volvera la mascara anterior.

   Lo que sucede básicamente es que cuando esta la mascara set el programa bloquea la señal pero la deja en espera y cuando se pone la máscara set, como la señal estaba en espera, esta se ejecuta y llama al handler.

6.

a) Cuando se manda la señal mientras está contando, se llama a la función handler_SIGALRM escribe por pantalla lo que tiene que escribir y sale del programa
b) Cuando se quita sigaction simplemente al mandarle la señal ejecuta el default de la señal por lo cual sale.

7.

a) Si que se podría modificar. La primera posición sería justo detrás del open ya que hasta que no acaban todas las llamadas al semáforo, este no se cierra.

8. a) Que se salta la espera de semwait() ya que recibe una señal saltandose el bloqueo de semwait()

b) Se queda en sem_wait por qué la señal se desecha por lo cual no tiene que atenderla y no sale de la espera.
c) Podríamos poner un bucle while que se asegurara de que el retorno de sem_wait es el esperado, y que si no lo es llamara de nuevo a sem_wait.

9.

A: Nada

B: sem_post(sem2);
   sem_wait(sem);

C:sem_post(sem2);
   sem_wait(sem);

D:sem_wait(sem2);

E: sem_post(sem);
sem_wait(sem2);

F: Nada

EJERCICIO DE CODIFICACIÓN

Lo primero que hacemos es comprobar que los argumentos introducidos son correctos. Después, crearemos las máscaras, handlers y semáforos que vamos a utilizar durante todo el programa.

Semaforos:

sem: Comprueba quien es el candidato

sem_vot: Comprueba si alguien está accediendo al archivo con votos.

Mascaras:

Estaremos bloqueando en todo momento SIGUSR1, SIGUSR2, SIGINT y SIGALARM. Menos en cierto momento en el que el candidato desbloqueará todas las señales para ver si ha recibido sigint de main.

Hace falta bloquear SIGINT y SIGALARM porqué si no las bloqueamos y se envia la señal sigint estas acuarán de forma inmediata y si se encontraran escribiendo en un archivo o con un archivo abierto, este no se cerraría nunca.

Manejadores:

SIGUSR1: Sirve para salir de un bloqueo

SIGUSR2: Sirve para salir de un bloqueo

SIGINT: Fin de ejecución

SIGALARM: Fin de ejecución

Funcionamiento:

Se crean los procesos hijos y se hace que llamen a una función en la que quedarán bloqueados a la espera de creación de todos los procesos. Una vez que se crean todos los procesos, main les envía SIGUSR1 para que se desbloqueen, lean la información de los procesos y continúen con su ejecución. En este momento los procesos pelearán por ver quien es el candidato, el primero en entrar en el semáforo se volverá candidato, el resto se volverán votantes y esperarán a la señal para votar.

Una vez reciban esta señal, abrirán un archivo, pondrán su respuesta y lo cerrarán de nuevo.

El proceso principal irá leyendo cada 0.001 segundos el archivo veririficando si se han escrito todas las respuestas necesarias, una vez que se han escrito todas las respuestas verá si ha sido o no aceptado y lo escribirá por pantalla, mandará señales al resto de procesos para continuar con la siguiente ronda y comenzarán a pelear por el candidato.

El resto de votantes una vez votan se quedan en el estado de pelear para ser candidato.

El proceso principal se quedará a la espera de recibir sigint o sigalarm y ejecutar la terminación del programa.

Analisis del problema:
Realizar un sistema de votaciones en el que un proceso es un candidato y el resto son votantes. Todo esto con la ayuda de semaforos y señales para la comunicación entre procesos.

Algoritmos utilizados:

Estructuras de datos.
Process_info:
Almacena los pids de los procesos creados asi como las votaciones de cada proceso.

Pruebas realizadas:
Para realizar pruebas hemos creado un script de bash que itera en un bucle llamando a el programa con diferentes argumentos. Aparentemente funciona sin ningún inconveniente.

Descripción de requisitos que satisface:

a) Es capaz de ejecutar con dos parametros, crear tantos procesos como especificados, y almacenar los pids y los votos en una estructura de datos. Además main también envia la señal SIGUSR1 a los votantes cuando el sistema está listo, y es capaz de manejar la señal SIGINT escribiendo lo que toca por pantalla. (Cumple todos los requisitos)
b) Los votantes se quedan a la espera de la señal de main, leen la información de un fichero binario, compiten por ser el candidato, se crea un candidato y el resto votantes, se realiza la votación a traves de un fichero binario en el cual se almacena el voto. El proceso candidato envia SIGUSR2 para especificar la señal que se envia. (NOTA: en el pdf se indica que candidato envie SIGUSR1 pero para no tener problemas con el SIGUSR1 enviado por main hemos decidido que era mejor idea enviar SIGUSR2). Nuestro proceso candidato alterna entre llamadas de 1ms lecturas al fichero para comprobar si se han escrito todos los votos. Cuando los procesos votantes reciben SIGUSR2 registran su voto en el fichero. Una vez se reciben todos los votos escribe el mensaje determinado según si es o no aceptado como candidato. Luego espera 250ms y comienza una nueva ronda.
c) Protección de zonas criticas: Hemos realizado esto mediante el bloqueo de señales con las máscaras y manejadores descritos al inicio.
d) Temporización: Hemos realizado un temporizador a través de una alarma con un manejador determinado para SIGALRM.
e) Analisis de ejecución: ¿Se produce algún problema de concurrencia? No debería de producirse ningún problema ya que todo está manejado mediante semáforos y señales. ¿Es sencillo o siquiera factible organizar un sistema de este tipo solo con señales? Sería una locura hacer este sistema solo con señales dado que los procesos tendrían que estar constantemente enviandose señales, quitando y poniendo máscaras y haciendo esperas.
f) Sincronización con semáforos: Nuestro sistema está implementado con semáforos.