Práctica 6: Comunicación y sincronización de hilos de distintos procesos

v1.0

1 Objetivos

El objetivo de esta práctica es afianzar nuestro conocimiento de los mecanismos de sincronización de procesos que ofrece un sistema POSIX y sus esquemas de uso. En la práctica haremos uso de semáforos y objetos de memoria compartida POSIX.

En esta práctica trabajaremos fundamentalmente con las funciones `shm_open`, `mmap`, `sem_open`, `sem_wait`, `sem_post`, `sem_close`, `sem_unlink`, `munmap`, `shm_unlink`, y `sigaction`, entre otras.

El archivo ficheros_p6.tar.gz contiene una serie de ficheros que pueden usarse como punto de partida para el desarrollo los ejercicios de esta práctica, así como unos makefiles que pueden ser usados para la compilación de los distintos proyectos.

2 Desarrollo de la práctica

Este ejercicio consiste en resolver el problema de la tribu de salvajes de la hoja de problemas de la asignatura, usando distintos procesos en lugar de hilos de un mismo proceso para simular cada uno de los salvajes y el cocinero. El enunciado del problema clásico es como sigue:

Una tribu de salvajes se sirven comida de un caldero con *M* raciones de estofado de misionero. Cuando un salvaje desea comer, se sirve una ración del caldero a menos que esté vacío. Si está vacío deberá avisar al cocinero para que reponga otras *M* raciones de estofado, y entonces se podrá servir su ración. Un cocinero y número arbitrario de salvajes se comportan del siguiente modo:

```
//Cocinero:
while (!finish) {
  putServingsInPot()
}

//Salvajes:
for (i = 0; i < NUMITER; i++){
  getServingFromPot()
  eat()
}
```

Se deben cumplir las siguientes restricciones:

- Los salvajes no pueden invocar `getServingFromPot()` si el caldero está vacío
- El cocinero sólo puede invocar `putServingsInPot()` si el caldero está vacío

Para simular este problema vamos a crear dos programas:

- Un programa *cocinero.c* que cree los recursos compartidos necesarios y luego ejecute la función `cocinero()`. El usuario sólo debe crear un proceso que ejecute este programa. Este programa registrará un manejador para las señales `SIGTERM` y `SIGINT`. Cuando sean capturadas el programa debe terminar, limpiando todos los recursos compartidos creados en el sistema.
- Un programa *salvaje.c*, que intente abrir los recursos compartidos que haya creado el programa cocinero. Si no los encuentra, avisará con un mensaje de error advirtiendo al usuario que debe ejecutar primero el programa cocinero. Una vez abiertos los recursos compartidos ejecutará la función `salvajes()`, que intentará comer `NUMITER` veces del caldero y después terminará. El usuario puede crear tantos procesos que ejecuten este programa como quiera, en distintos terminales o en el mismo terminal lanzando los procesos en *background* (por ejemplo puede usar un bucle for de bash y lanzar varios procesos que ejecuten este programa).

Se aconseja a los estudiantes utilizar semáforos con nombre para la sincronización de los distintos procesos y una región de memoria compartida para alojar la variable que representa el contenido del caldero.

Las funciones `putServingsInPot()`, `getServingsFromPot()` y `eat()`, además de implementar la sincronización necesaria deben mostrar por el terminal menajes que permitan comprobar el funcionamiento del programa, indicando en cada mensaje el id del proceso correspondiente.