Info_sw
www.wuolah.com/student/Info_sw

APSO_apuntes_practica4.pdf

APSO. Parte 2: programación

2° Administración y Programación de Sistemas Operativos

Grado en Ingeniería Informática

Escuela Técnica Superior de Ingeniería
UHU - Universidad de Huelva

APSO. PROGRAMACIÓN DEL SISTEMA.

APUNTES PRÁCTICA 4:

SINCRONIZACIÓN PADRE-HIJO (en lugar de hacer sleep(x) como hacíamos en prac3)

wait() : llamada al sistema que ejecuta el proceso padre.

El proceso padre espera hasta que uno de sus hijos muera (no podemos especificar cuál de los

hijos debe esperar a que muera).

wait() devuelve un número entero. Este número devuelto será el pid del hijo que muere. Si quiero

esperar a que un determinado hijo muera puedo repetir este proceso hasta que wait() devuelva el

pid del hijo que quiero esperar

exit(int valor): lo lanza el hijo. Para que un hijo termine y pueda mandarle un mensaje al padre

usando el parámetro. Nosotros no haremos uso del parámetro, lo usaremos siempre sin parámetros.

********** SINCRONIZACIÓN MEDIANTE SEÑALES **********

Habrá un proceso que emitirá una señal y otro proceso que recibirá la señal. Cuando el segundo

proceso reciba una señal, este parará su ejecución por la línea de código que esté en ese momento

y tratará la rutina de atención de la interrupción o driver (nosotros implementaremos esto mediante

funciones). Posteriormente, el proceso continúa su ejecución por donde lo había dejado.

EL proceso que debe recibir la señal debe estar preparado antes para poder recibirla. Si el proceso

no está preparado para recibir una determinada señal, este proceso MUERE.

sigaction(señal,param2,NULL) -> Prepara a un proceso para que pueda recibir la señal x

Ejemplo: sigaction(10,...,NULL) -> prepara al proceso para recibir la señal 10

El segundo parámetro es una variable de tipo "struct sigaction". Rellenamos los dos campos del

struct: "sa_flags" a 0 y en el campo "sa_handler" el nombre de la rutina o funcion de tratamiento

de la rutina. -> ver ejemplo en apuntes página 4.

signal(señal,nombre_funcion) -> idéntico que el sigaction(...).

La diferencia es que el signal prepara al proceso para recibir una señal una sola vez. Cuando el

proceso vuelva a recibir esa señal, el proceso no estará preparado para recibirla, y morirá.

Actualmente el signal también prepara al proceso para recibir una señal muchas veces.

Pero si usamos el signal, será conveniente también ponerlo en la primera línea del código de la

función de tratamiento de la rutina porque puede haber sistemas más antiguos que no tengan esta

modificación del signal implementada.

kill(int pid,señal). //EMITIR o ENVIAR UNA SEÑAL A OTRO PROCESO. Para ello hay que conocer el pid del

proceso al que envío una señal.

Ejemplo: kill(1090,10); -> envía al proceso de pid 1090 la señal 10

alarm() -> es una alarma. Un proceso que en su código haga un alarm(nº segs) recibirá la señal 14

cada x segundos (parametro). El proceso debe estar preparado por tanto para recibir la señal 14

y debemos tener una funcion para tratar la señal 14. Esta función la usaremos en la práctica final

para que cuando un barco esté esperando demasiado tiempo se aburra y se vaya)

-Si hago un alarm(5) el proceso que la hace recibirá la señal 14 cada 5 segs.

-Si más tarde hacemos un alarm(6), el alarm(5) se anula y la señal 14 se recibirá cada 6 segs.

Con el alarm(), el proceso no se para, sino que continúa su ejecución de código.

-Si hago un alarm(0) anulamos la recepción de la señal 14.

pause() -> hace que un proceso se espere hasta que llegue una señal. No podemos decirle para qué

señal queremos esperar.

sleep() -> ya sabemos lo que hace: espera una cierta cantidad de segundos.

Ojo que si durante un sleep llega una señal, se ejecutará la rutina de esa señal y posteriormente se

salta los segundos que queden del sleep().