Instituto Universitario Aeronáutico

Sistemas Operativos

Procesos

Objetivo

---------

En este práctico se introducen llamadas al sistema operativo

que nos permiten manejar procesos, su creación, finalización

y señalización.

Se escribirán programas que nos permitan utilizar las llamadas

siguientes:

\* fork() Esta llamada nos permite crear nuevos procesos

\* wait() Esta llamada nos permite esperar (wait) la finalización

de un proceso dado.

\* kill() Nos permite enviar una señal a un proceso determinado

\* signal() Nos permite capturar una señal enviada a un proceso.

Las anteriores son descripciones básicas, para mas detalles

consulte las páginas de manual.

Se espera que respete las sugerencias en cuanto estilo de codificación

planteadas en la Practica 1.

Práctica Cree un directorio en su $HOME y trabaje allí

1. Practique el comando ps para visualizar procesos y su jerarquia,

y el comando kill para el envio de señales.

$ ps aux

$ ps al

$ kill <signal> PID

El primer comando nos permite obtener información sobre todos los

procesos en ejecución, el segundo nos permite ver los procesos en

su jerarquía padre-hijo, el tercero nos permite enviar una señal al un proceso determinado.

2. Escriba un programa (en C) que utilice la llamada fork() para

crear un nuevo proceso. Es necesario que cada proceso escriba un

mensaje diferente en pantalla.

Utilice el siguiente esquema de programa

int main() {

int pid;

...

pid = fork();

if (pid == 0) {

/\* hijo \*/

...

} else {

/\* padre \*/

...

}

return 0;

}

Estudie la página de manual de la llamada fork() .

3. Escriba un programa similar al anterior, de modo que cada proceso

obtenga su propio numero de proceso y el de su padre; a continuación

imprima el resultado. Estudie y utilice las llamadas getpid() y getppid().

4. Escriba un programa en el cual creamos un proceso y configuramos

al proceso “padre” para que espere, por medio de la llamada wait() ,

a que termine su hijo antes de proseguir, luego de lo cual debe

reportar el status de salida del proceso hijo.

Use la llamada wait() . Se la debe invocar como

...

wait(&STATUS);

...

Use el status de salida, analícelo por medio de las funciones macro

definidas en la página de manual de la llamada wait() .

5. Escriba un programa similar al anterior, pero modificado de modo

que el hijo quede en espera indefinida, luego desde una terminal envíe

una señal (por medio del comando kill para que termine y observe el

comportamiento del status de salida.

Investigue las señales posibles por medio de man 7 signal .

6. Escriba un programa que capture las señales “Ctrl-C” (Interrumpir)

y “Ctrl-Z” (Suspender), en el primer caso el programa debe escribir un mensaje en pantalla, en el segundo debe terminar.

Utilice el siguiente procedimiento: defina una función que retorne

void para manejar la señal y asigne una señal a esta función por medio

de la llamada signal()

void sigmanager(int sig\_num) {

/\* Reasignar la señal \*’/

signal(SIGNAL, sigmanager):

...

fflush(stdout);

}

...

signal(SIGNAL, sigmanager);

7. Escriba un programa que cree un hijo, capture las mismas señales

del programa anterior, se las reenvie al hijo y reproduzca la

funcionalidad previa, es decir que el hijo escriba un mensaje para

la primer señal y termine para la segunda, al terminar el hijo también

debe terminar el padre.

8. Haga un push de su carpeta de trabajo