**Ejercicios AL, 19feb2019**

**Programas concurrentes (padre-hijos-nietos).**

**OBJETIVO.**

Conocer los diferentes formas en que se arranca un proceso (padre o hijo), vía scripts, vía línea de comando o a través de los APIs (funciones, métodos) del lenguaje de programación (C++); tal y como lo hace el sistema operativo.

En el caso de los APIs de programación se emplean tanto la cómoda, cara, y poco óptima función ***system( )*** así como las tradicionales funciones básicas ***fork()*** y la familia ***exec…(…)***.

**INDICACIONES SOBRE EL DESARROLLO**

Lleve a cabo las revisiones y desarrollos de programas en C++. En todos los puntos que sigue tendrá que explicar con detalle cómo logró llevar a cabo lo pedido, indicando trayectorias, comandos con su despliegue y / o acciones realizadas. En el caso de despliegue de comandos explique el significado de lo desplegado. Cuando haya preguntas específicas, no basta con aplicar y desplegar el resultado del comando, deberá usted contestarlas.

**DESARROLLO**

En el archivo ***CHeaders.xlsx*** puede usted encontrar algunos de los Headers de C++ para Linux/Unix. Para el caso de ***fork()*** y la familia ***exec…(…)*** lea el archivo ***Fork&Execfam.pdf***.

Dada la situación de ejecución de programas en su directorio de trabajo es fundamental que definan él “.” (punto) al inicio del PATH; esto se logra ejecutando en la terminal directamente el comando **“setenv PATH .:$PATH”** o mediante el comando ***source ~/.midir***, que se encuentra en su *home directory sisops*, que lo construimos en clases pasadas.

1. Ejecución de un nuevo programa, dentro de un proceso, con ***execlp()***. En el fólder se encuentran los archivos *subs.cc* y *otro01.cc*; deles rápidamente una mirada y después compile ambos programas para construir ambos ejecutables de *subs.exe* y *otro01.exe*.

En primer lugar ejecute *otro01.exe HOLA*, indicando lo que representa cada uno de los valores impresos.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Dentro de *subs.cc* se encuentra el API ***execlp( file, argv0, argv1, , , argvN, (char \*) 0 )***; *file* representa el nombre del archivo del programa ejecutable ejecutable, ***argv0, argv1, , , argvN*** son los argumentos que el sistema

Ahora, ejecute el programa *subs.exe*

**¿Cómo se da cuenta de que el programa del proceso ha sido substituido? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**¿Cuál es PID del proceso padre de *subs.exe*?\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**¿El pid de *subs* es idéntico o diferente que el de *otro01*? ¿Sí o no, por qué? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**¿Cuál es PID del proceso padre de *otro01.exe*? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**¿Cómo es que *otro01.exe* pudo imprimir el texto “... recibiendo *viajarX*? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ explique.**

**¿En el programa *subs.exe* hay alguna instrucción que nunca se llegue a ejecutar? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

1. Creación de procesos hijos con ***fork()***. En este fólder se encuentran los archivos *padre.cc* e *hijofinal.cc*. *padre.cc* contiene el programa principal (*main*) e *hijofinal.cc* contiene la operatividad del hijo.

El API *fork()* se encuentra en el archivo *padre.cc* para poder crear un proceso clon hijo. Puede regresar cualquiera de tres valores ( **<0**: error; **=0**: el proceso hijo recibe este resultado; **>0**: el proceso padre recibe este resultado equivalente al PID del proceso hijo).

Siempre es conveniente que siempre se compilen ambos programas, después de modificarl ambos o alguno de ellos, para esto se les provee con el archivo *pahi.src* que se debe aplicar como *source pahi.src*, logrando así los ejecutables *padre.exe* e *hijofinal.exe*.

Ejecute el programa *padre.exe* y tómese unos minutos para analizar lo que despliega el programa en ejecución y porque lo hace; vaya comparando contra los códigos de *padre.exe* e *hijofinal.exe*.

Conteste a las siguientes preguntas:

**¿Una vez creado el proceso *clon hijo* (copia del padre), que es lo que devuelve como resultado la función *fork()*, respectivamente, en los procesos *padre* y *clonhijo*.**

En el proceso padre, el objetivo de la función

*wait( &childstts );*

es lograr que la señal SIGCLD enviada por el proceso hijo (a partir de *exit(status)* ) sea "atendida" mediante *wait* que toma del sistema el *status* de terminación enviado por el proceso hijo. Si la función no es ejecutada entonces el proceso hijo se convierte en proceso zombie.

El API *wait( &childstts )* puede regresar cualquiera de tres valores ( **<0**: error; **=-1**: no tiene ningún proceso hijo; **>0**: PID del proceso hijo).

En la instrucción

childpid = *wait( &childstts );*

**¿Para qué es útil conocer el valor devuelto por la función *wait* y que queda en *childpid*?\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**¿Cuándo sería realmente útil?\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**¿Por qué se manda ejecutar el comando "ps -l" tanto desde el proceso *clon hijo* como desde *hijofinal* y no conviene hacerlos desde el proceso *padre*? ¿Desde dónde se verían más procesos con "ps -l"?\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**¿Hay alguna instrucción dentro de *padre.cc* que nunca se llegue a ejecutar?\_\_\_\_\_\_\_\_\_\_**