Ejercicios BGE
Programas concurrentes (padre-hijos-nietos).

OBJETIVO.
Conocer las 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***.

1) Primero entre a la máquina virtual con Ubuntu.

2) Ahora vamos a instalar la herramienta *g++*. El profesor le indicara como hacerlo y probar si la instalación se llevó a cabo.

3) También el profesor le indicará como llevar a cabo la compilación de programas C y su respectiva ejecución.

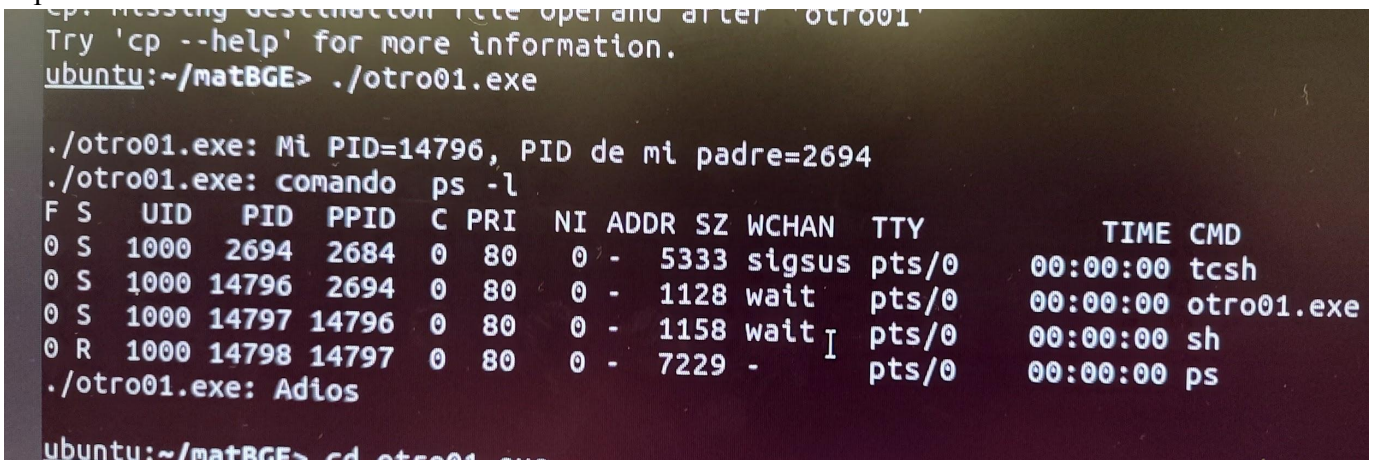
4) Creación del archivo "*.mydot*" en su *home directory sysops*.
   Dada la situación de ejecución de programas en su directorio de trabajo es fundamental que definan él **"."** (punto) al inicio del *PATH*.
   El archivo "*.mydot*" lo deberá crear en el directorio base, y deberá contener el comando ***setenv PATH ".:$PATH"***, para que pueda rápidamente agregar el directorio de trabajo en el PATH de su terminal.
   Después, desde cualquier terminal, y desde cualquier directorio, podrá aplicar el comando:
   source $home/.mydot, agregándose el directorio de trabajo al PATH.
   Crear archivo gedit .mydot
   En el archivo .mydot agregar
   Setenv PATH ".:$PATH"

5) 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*.

g++ otro01.cc esto manda a nombrar y crea a.out

g++ otro01.cc –o otro01.exe y crea el ejecutable .exe

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



PID es el ID del proceso

PPID es el ID del proceso padre

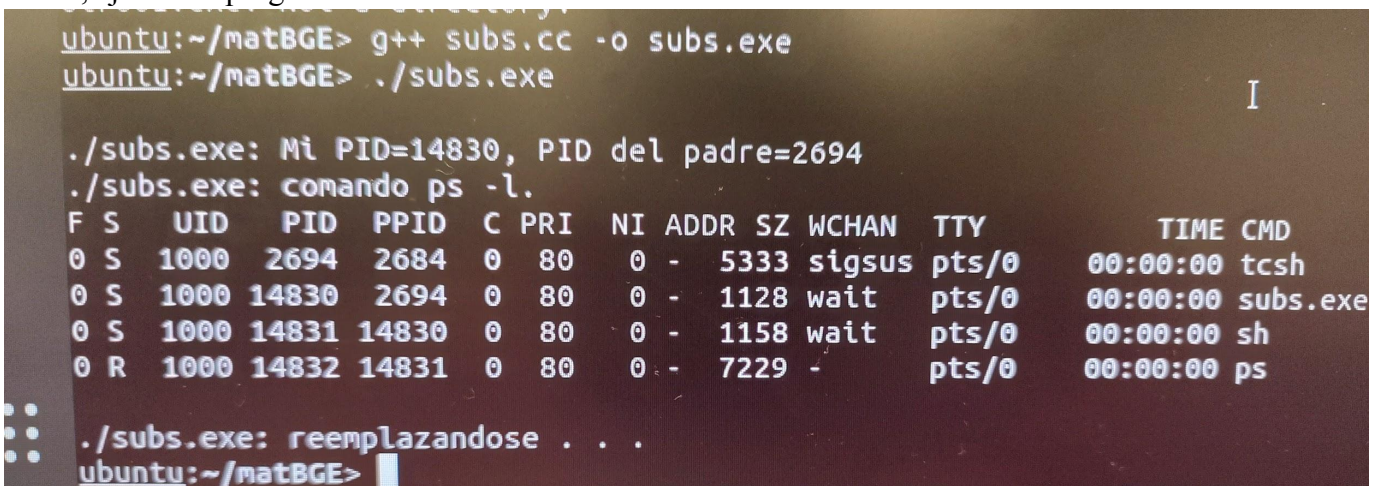
SZ tamaño en páginas físicas de la imagen principal del proceso.

WCHAN es el estado de ejecución del proceso

CMD el comando con todos sus argumentos como string

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? **Por el cambio del PID del segundo proceso, y por el cambio de nombre en el CMD**

¿Cuál es PID del proceso padre de *subs.exe*? **tcsh con el PID 2694**

¿El pid de *subs* es idéntico o diferente que el de *otro01*? ¿Sí o no, por qué? **Es diferente porque se trata de un .exe distinto, del encapsulamiento o instancia de un programa diferente.**
¿Cuál es PID del proceso padre de *otro01.exe*? **El mismo que subs.exe, es decir el PID 2694**
¿Cómo es que *otro01.exe* pudo imprimir el texto "... recibiendo *viajarX*? **Porque se sustituyó el programa por una nueva ejecución y en esa nueva ejecución se pasó como parámetro "viajarX"**
¿En el programa *subs.exe* hay alguna instrucción que nunca se llegue a ejecutar?
return 0, no se ejecuta lo que hay después del execlp(...)

6) 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 modificar 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 devuelve los PID correctos del padre y del hijo, mientras que en el proceso clonhijo se devuelve 2 procesos nombrados padre, que después se nombrará como hijofinal con el mismo PID anterior de un proceso "padre".

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*?
Es el código del status que resultó de que el proceso hijo se ejecutará. Es útil para saber si hubo un error o fue exitoso.
En childpid queda el PID de mi hijo.
¿Cuándo sería realmente útil? **Sirve cuando tengo muchos hijos y quiero saber su PID y si se han ejecutado con éxito para tomar acciones posteriores.**

¿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"? **Porque el proceso clon hijo recibió el childpid = 0 y dentro del if se ejecuta el comando "ps -l" con system. A su vez, el hijofinal sustituye al clonHijo y por eso se vuelve a ejecutar el "ps -l".**
No conviene porque no se observaría que el padre.exe se sustituyó por el hijofinal.exe

En general, se verían más procesos desde el hijo que esté más abajo en la llamada para crear hijos.

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

No, porque el padre no ejecuta el execlp(...)