1. Responda en forma sintética sobre los siguientes conceptos
   1. Programa y proceso

Un programa es la totalidad de instrucciones compiladas o de un lenguaje interpretado residente en memoria secundaria y que hacen al comportamiento de una aplicación, junto con datos asociados

Un proceso es una instancia de un programa en ejecución, que junto con instrucciones, datos y estructuras de control, tienen una tarea específica a cumplir y que residirá en memoria principal como tal hasta cumplirla

* 1. Defina Tiempo de retorno (TR) y Tiempo de espera (TE) para un Job
* Tiempo de retorno: es el tiempo total que tarda un proceso determinado desde que es cargado efectivamente en memoria hasta que concluye su tarea en el CPU
* Tiempo de espera: tiempo de existencia de un proceso durante el cual no está haciendo uso del CPU. Es básicamente la diferencia entre el tiempo de retorno y el tiempo de ocupación del procesador
  1. Defina tiempo promedio de retorno (TPR) y tiempo promedio de espera (TPE) para un lote de *Jobs*
* TPR: promedio de tiempos de retorno entre todos los procesos pertenecientes al lote. Se calcula como la suma del TR de cada proceso, dividida entre la cantidad de procesos
* TPE: promedio de tiempos de espera calculando entre todos los procesos involucrados en el lote. Se calcula como la suma del TE de todos los procesos, dividida entre la cantidad de procesos
  1. ¿Qué es el *quantum?*

Es el tiempo de procesador que se le asigna a los procesos en el algoritmo *round robin***.** Determina la longitud máxima de la ráfaga del procesador que podrá usar cada proceso al entrar al CPU, puede ser más corta si es que el proceso concluye su tarea o debe esperar algún otro recurso antes de acabar la ráfaga

* 1. ¿Qué significa que un algoritmo de *scheduling* sea apropiativo o no apropiativo (*preemptive/ non-preemptive)*?
* Apropiativos: los procesos pueden ser sacados de la CPU según disparadores externos al propio proceso. En estos casos el SO fuerza la salida de un proceso del procesador para asignarle el uso a otro, independientemente de si el proceso retirado concluyó o no su uso del CPU
* No apropiativos: aquellos en los que los procesos no abandonan el CPU más que por condiciones internas del proceso, como haber concluido su uso o requerir la conclusión de una operación de E/S
  1. ¿Qué tareas realizan?
* Short term scheduler: selecciona un proceso de los estado *ready* para pasarlo a ser ejecutado
* Long term scheduler: admite procesos, de estado *nuevo* a *listo*
* Medium term scheduler: se ocupa del *swap out* de la memoria de el o los procesos que sea necesario para mantener el equilibrio del sistema. También se ocupa de la carga en memoria principal
  1. ¿Qué tareas realiza el dispatcher?

Cambia el contexto de los procesos, descarga los registros del procesador y otros datos relacionados a las estructuras de datos que se usan para controlar el proceso saliente

1. Procesos
   1. Investigue y detalle para qué sirve cada uno de los siguientes comandos

* top: provee una vista dinámica y en tiempo real del sistema en ejecución. Puede tanto mostrar información del sistema en general, como una lista de tareas que están siendo administradas por el kernel. Los tipos de información mostrados en ambos espacios (información del sistema y de tareas) pueden configurarse hacerse persistentes entre reinicios del sistema. El programa incluye una interfaz interactiva limitada para la manipulación de procesos
* htop: visualizador de procesos similar a top, pero permite recorrer la ventana de forma vertical y horizontal, de modo que se pueden ver todos los procesos ejecutándose en el sistema, junto con sus líneas de comando completas, de la misma forma que verlos como un árbol de procesos, seleccionando varios procesos y trabajado sobre ellos a la vez. Las tareas relacionadas con los procesos (terminarlos, modificar su nice number), se pueden llevar a cabo sin indicar sus PIDs.
* ps: lista los procesos en ejecución
* pstree: muestra los procesos en ejecución como un árbol. El comando une ramas idénticas colocándolas entre corchetes y prefijando el contador de repeticiones (ejemplo: padre---3\*[proceso]). Los hilos hijo se muestran bajo el proceso padre y se indican con el nombre del proceso padre entre llaves (ejemplo: padre---8\*[{padre}]). Al final de la declaración de opciones puede indicarse un pid, en cuyo caso se tomará como proceso raíz del árbol, o un usuario, con lo que se mostrarán todos los árboles cuya raíz sea un proceso del que el usuario es propietario. Si no se indica ninguna de las opciones, se muestra un árbol con init o systemd como raíz.
* kill: envía la señal especificada al o a los procesos indicados. Si no se especifica una señal, se envía TERM, que eliminará los procesos que no puedan capturarla. La sintaxis es kill [-s señal | -p] pid… o kill

-l [ señal ]

* pid puede ser:
  + Un número natural mayor a 0, en cuyo caso se señalará al proceso con ese PID
  + 0: en cuyo caso todos los procesos en el grupo de procesos actual serán señalados
  + -1: todos los procesos con PID mayor a uno serán señalados (salvo el propio kill)
  + -n, siendo n un número mayor a 1: todos los procesos en el grupo de procesos n son señalados. Si se usa un argumento de este tipo, se debe especificar la señal primero, o se debe anteponer --, de otra forma -n será interpretada como la señal a enviar
* pgrep: busca todos los procesos actualmente en el sistema cuyas características coincidan exactamente con las indicadas en la línea de comandos, y lista su PID en pantalla.

La sintaxis es pgrep [-flvx] [-d delimiter ] [-n|-o] [-P ppid ,...] [-g pgrp ,...] [-s sid ,...] [-u euid ,...] [-U uid ,...] [-G gid ,...] [-t term ,...] [ pattern ]

* + -d delimitador : establece la cadena que se utilizará para delimitar cada PID en la salida, por defecto es un salto de línea (exclusivo de pgrep ).
  + -f : el patrón pattern normalmente sólo se compara con el nombre del proceso, si se establece -f se comparará con la línea de comandos completa.
  + -g pgrp , ... : sólo tomará procesos que se correspondan con los PGIDs indicados. El PGID 0 es interpretado como el PGID bajo el que se ejecuta el propio comando.
  + -G gid , ... : sólo tomará procesos cuyo ID real de grupo coincida con uno de los indicados. Se pueden utilizar valores numéricos o simbólicos.
  + -l : lista el nombre del proceso además de su PID. (exclusivo de pgrep ).
  + -n : sólo selecciona el más nuevo de los procesos encontrados (el que haya iniciado más recientemente).
  + -o : el opuesto a -n, selecciona al más antiguo de los procesos encontrados
  + -p ppid , ... : sólo selecciona procesos cuyo PPID ( parent process ID ) coincida con uno de los indicados.
  + -s sid , ... : sólo selecciona aquellos procesos cuyo ID de sesión de proceso coincide con uno de los indicados. El ID de sesión 0 es interpretado como aquel bajo el que se ejecuta el propio comando.
  + -t term , ... : sólo selecciona procesos cuya terminal de control se encuentra entre las indicadas. El nombre de la terminal debe ser especificado con el prefijo “/dev/”.
  + -u euid , ... : sólo selecciona procesos cuyo ID de usuario efectivo coincide con el indicado. Pueden utilizarse tanto valores simbólicos como numéricos.
  + -U uid , ... : sólo selecciona procesos cuyo ID de usuario real coincide con el indicado. Pueden utilizarse tanto valores simbólicos como numéricos.
  + -v : realiza la negación de la comparación (selecciona los que no respetan el/los patrones)
  + -x : sólo selecciona aquellos procesos cuyo nombre coincide exactamente con el patrón indicado (o su línea de comando de origen si se selecciona la opción -f )
* pkill: busca todos los procesos cuyas características coincidan exactamente con las indicadas en la línea de comandos, y les envía el mensaje indicado en el llamado al comando. Si no se indica una señal, se enviará la predeterminada SIGTERM.

La sintaxis es pgrep [ -signal ] [-fvx] [-n|-o] [-P ppid ,...] [-g pgrp ,...] [-s sid ,...] [-u euid ,...] [-U uid ,...] [-G gid ,...] [-t term ,...] [ pattern ]

Comparte las opciones con el comando pgrep , con la excepción de:

- signal : define la señal a ser enviada a cada uno de los procesos que coincidan con los patrones indicados. Pueden utilizarse indistintamente valores simbólicos o numéricos de señal.

(y aquellas indicadas en pgrep como exclusivas de dicho comando)

* killall: envía la señal especificada al o a los proceso indicados por nombre. Si no se especifica una señal, se envía TERM, que eliminará los procesos que no puedan capturarla. La sintaxis es kill [-s señal | -p] nombre… o kill -l [ señal ] . Si el nombre de comando indicado no es una expresión regular (opción -r) y contiene /, el proceso ejecutando ese archivo será seleccionado para morir independientemente de su nombre
* renice: modifica la prioridad de uno o más procesos en ejecución. El primer argumento es el valor de prioridad ( nice number ) a ser utilizado. Los demás argumentos son identificados como PIDs (la opción por defecto), PGIDs, UIDs o nombres de usuario. Ejecutar renice sobre un grupo de procesos (por PGID, UID o nombre de usuario) cambia la prioridad de planificación de todo el grupo involucrado.

La sintaxis es renice [ -n ] priority [ -g | -p | -u ] identifier ...

Opciones:

-n , --priority : especifica la prioridad a ser asignada al proceso, grupo de procesos o usuario. Indicar -n o --priority es opcional, pero si se lleva a cabo debe ser el primer argumento

-g , --pgrp | -p , --pid | -u , --user : aplica la prioridad indicada a los procesos cuyo PGID, PID (la opción por defecto), o usuario (indicado con nombre o UID) coincida con los listados a continuación de la opción.

* xkill: fuerza la desconexión de un X server con uno de sus clientes (básicamente permite cerrar ventanas de programas). Forzar la desconexión no implica necesariamente que los procesos asociados al cliente terminarán de forma correcta, o siquiera que serán terminados.

La sintaxis es xkill [-display displayname ] [-id resource ] [-button number] [-frame]

[-all]. Si no se indica ninguna de las opciones, se mostrará un cursor en forma de x (o una calavera, según la distribución y conjunto de símbolos del sistema) con el que se podrá seleccionar el cliente (ventana) a cerrar

* atop: es un monitor interactivo de la carga de trabajo en un sistema operativo Linux. Indica la ocupación de los recursos más críticos de hardware - desde una perspectiva de rendimiento - a nivel de sistema (i.e. CPU, memoria, disco y red). También muestra qué procesos son responsables de la carga indicada de cpu y memoria. La carga de disco y red dependen de la presencia de configuraciones y módulos del kernel. Cada un determinado intervalo (predeterminado: 10 segundos, pero se puede modificar en la línea de comandos) se muestra información sobre la ocupación de recursos del sistema, seguida por una lista de procesos que han estado activos durante el último intervalo (si algún proceso no fue modificado durante este intervalo, no será mostrado, salvo que se haya presionado la tecla ‘a’). Si la lista de procesos activos no entra en la pantalla, se mostrará todos los que entren, ordenados en función de su actividad. Al igual que top, los intervalos se repiten hasta alcanzar el número de muestras indicadas en el comando de llamada al programa, o hasta que se presione la tecla 'q' en caso de estar en modo interactivo.

Cuando se inicia, el programa evalúa si la salida estándar está conectada con una pantalla con un archivo o pipe. En el primer caso produce códigos de control de pantalla, y se comporta de forma interactiva; en el segundo caso produce una salida de texto plano en formato ASCII.

En modo interactivo, la salida de atop es escalada de forma dinámica en función de las dimensiones actuales de la ventana o pantalla. Si se modifica el ancho de la pantalla o ventana, se agregarán o quitarán columnas de manera automática, mostrando aquellas más importantes que entren en la dimensión actual (para este fin, cada columna tiene asignado un valor de relevancia)

* 1. Observe detenidamente el siguiente código. Intente entender lo que hace sin necesidad de ejecutarlo

#include <stdio.h>

#include <sys/types.h>

#include <unistd.h>

int main ( void ) {

int c ;

pid\_t pid ;

printf ( "Comienzo . : \ n " ) ;

for ( c=0 ; c<3 ; c++ )

{

pid = fork ( ) ;

}

printf ( " Proceso \n " ) ;

return 0 ;

}

¿Cuántas líneas con la palabra “Proceso” aparecen al final de la ejecución de este programa? 8

* 1. Vamos a tomar una variante del programa anterior. Ahora, además de un mensaje, vamos a añadir una variable y, al final del programa vamos a mostrar su valor. El nuevo código del programa se muestra a continuación.

#include <stdio.h>

#include <sys/types.h>

#include <unistd.h>

int main ( void ) {

int c ;

pid\_t pid ;

printf ( "Comienzo . : \ n " ) ;

for ( c=0 ; c<3 ; c++ )

{

pid = fork ( ) ;

}

p++

printf ( " Proceso %d\n " ) ;

return 0 ;

}

¿Qué valores se muestran por consola? Proceso 1\*8

¿Todas las líneas tendrán el mismo valor o algunas líneas tendrán valores distintos? Todas tendrán el mismo valor

¿Cuál es el valor (o valores) que aparece?. Ejecute el programa y compruebe si su respuesta es

correcta, Modifique el valor del bucle for y el lugar dónde se incrementa la variable p y compruebe los nuevos resultados

* 1. Comunicación entre procesos

Un pipe o canal es un búfer circular que permite que dos procesos se comuniquen entre sí siguiendo el modelo productor-consumidor. En sentido de implementación, es un seudo archivo que reemplaza la salida estándar de un proceso y la entrada estándar de otro, de forma tal que puedan enviar y recibir mensajes respectivamente. Funciona como una cola de tipo FIFO, en la que las sucesivas escrituras se van leyendo por orden de llegada.

Cuando se crea una tubería, se le establece un tamaño fijo en bytes. Cuando un proceso intenta escribir en la tubería, la petición de escritura se ejecuta inmediatamente si hay suficiente espacio; en caso contrario, el proceso se bloquea. De manera similar, un proceso que lee se bloquea si intenta leer más bytes de los que están actualmente en la tubería; en caso contrario, la petición de lectura se ejecuta inmediatamente. El sistema operativo asegura la exclusión mutua: es decir, en cada momento sólo puede acceder a una tubería un único proceso.

Hay dos tipos de tuberías: con nombre y sin nombre. Sólo los procesos relacionados pueden compartir tuberías sin nombre, mientras que los procesos pueden compartir tuberías con nombre tanto si están relacionados como si no. Por último, un proceso puede enviar una señal de error POSIX SIGPIPE , que indica que recibió un pipe que no tiene lectores.

En C los pipes se crean a través de la instrucción pipe(&fd[N]) con N = 0 o N = 1 y fd un arreglo de dos enteros

Para la creación de un parámetro se debe establecer como parámetro un arreglo de dos enteros, indicando el primer dato ([0]) si el proceso recibirá datos a través del canal (es decir que se convierte en el descriptor de archivo de su entrada estándar), y el segundo dato si utilizará el pipe para escribir (convirtiéndolo en el descriptor de archivo de su salida estándar).

Para lograr una comunicación correcta, los procesos involucrados deberán cerrar el extremo del canal que no les incube ( close(&fd[0]) si el pipe fue abierto con pipe(&fd[1]) y viceversa. De otra forma, el pipe nunca se dará por cerrado, puesto que no aparecerá una marca de EOF

Los canales o tuberías sólo permiten comunicación de una sola vía, es decir que un proceso sólo puede escribir o leer un pipe, pero no ambas a la vez

* 1. ¿Cuál es la información mínima que el SO debe tener sobre un proceso?¿En qué estructura de datos asociada almacena dicha información?
  2. ¿Qué significa que un proceso sea “CPU Bound” y “I/O Bound”?
* CPU Bound : significa que el proceso requiere principalmente de tiempo de procesador para su ejecución
* I/O Bound : significa que el proceso depende en gran medida de operaciones de E/S para su ejecución
  1. ¿Cuáles son los estados posibles por los que puede atravesar un proceso?
* Nuevo (new): implica la carga de datos del proceso en memoria, incluyendo las estructuras asociadas (PCB, etc.). Esta etapa se considera concluida una vez que termina de cargarse en memoria, y se considera admitido por el sistema.
* Listo para ejecutar (ready/ready to run): según las prioridades establecidas para el proceso, deberá o no esperar a que su primera instrucción sea cargada en el CPU. En este estado el proceso compite con otros por el uso del procesador.
* Ejecutándose (running): es la ejecución normal de las instrucciones involucradas en el proceso (luego de haber sucedido una selección y el correspondiente context switch ). De este estado pueden suceder tres caminos: por un lado darse la finalización normal de sus instrucciones; otra opción es que sea interrumpido, retornando al estado ready .
* Esperando su ejecución/bloqueado (waiting): si el proceso requiere uso de algún recurso E/S, o por algún motivo se hace esperar, hasta que el tiempo necesario no haya transcurrido., las instrucciones subsiguientes a él no podrán ejecutarse. Por este motivo el proceso es puesto en espera, sin posibilidad de ser cargado en el CPU. Una vez concluida la E/S o pasado el tiempo necesario para que se pueda retomar el proceso, este es cargado nuevamente en la cola de ejecuciones pasando al estado ready
* Finalizado/saliente (terminated): es más que nada la eliminación de memoria principal de todas las estructuras asociadas, finaliza con el borrado de la PCB.
* Suspendido: es un estado que puede superponerse con listo y bloqueado, implica que el proceso en cuestión, más allá de su estado en cuanto a su capacidad intrínseca de acceder al procesador, no se encuentra cargado en memoria principal, si no exclusivamente en memoria secundaria
  1. Explique mediante un diagrama las posibles transiciones entre los estados
  2. ¿Qué scheduler de los mencionados en 1) f. se encarga de las transiciones?

De las transiciones del estado listo a ejecutándose, se encarga el planificador de corto plazo

De las transiciones a y desde el estado suspendido se encarga el planificador de mediano plazo

De la admisión y transición al estado listo se encarga el planificador de largo plazo