Departamento de Cs. e Ingeniería de la Computación
Universidad Nacional del Sur

Sistemas Operativos y Distribuidos

Trabajo Práctico N° 3

Procesos e Hilos

Segundo Cuatrimestre de 2015

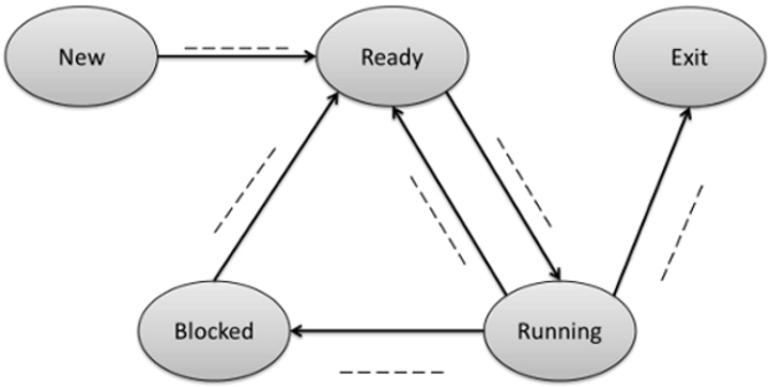
Ejercicios

1. ¿Qué entiende por *proceso* y *contexto de ejecución*?

2. Describa al menos cinco responsabilidades básicas que debería tener un sistema operativo con respecto a la administración y manejo de procesos.

3. Indique al menos cuatro razones que conlleven a la creación y a la finalización de un proceso.

4. Considerando el siguiente diagrama de estados



   a) Defina y describa cada estado.

   b) Determine la etiqueta de cada arista y la razón de cada transición.

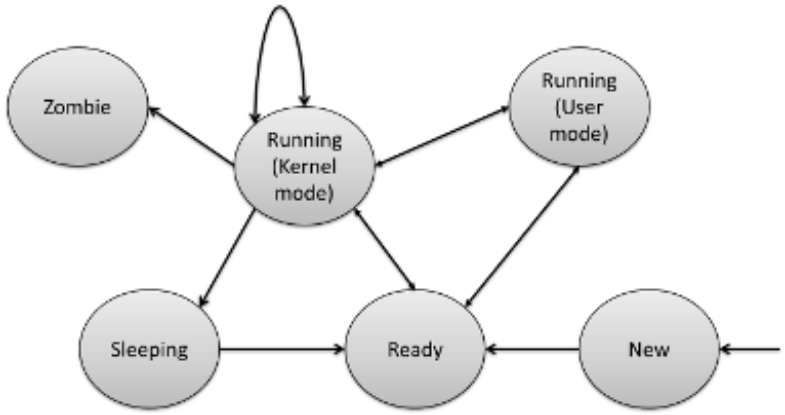
5. Describa el número mínimo de acciones que debería llevar a cabo un sistema operativo al crear un nuevo proceso.

6. ¿Cuál es el objetivo de la utilización por parte del sistema operativo de la estructura de datos típicamente conocida como *Process Control Block (PCB)*? ¿Cuál es el número mínimo de elementos que permitirían caracterizar unívocamente a un proceso durante su ejecución?

7. Especifique las acciones llevadas a cabo por el *kernel* al realizar un cambio de contexto entre procesos.

8. Describa el mecanismo tradicional de creación de procesos implementado por los sistemas operativos de la familia UNIX. En particular, considere el esquema utilizado por el sistema operativo GNU/Linux. ¿Qué llamadas al sistema intervienen en el proceso?

9. A partir del siguiente diagrama de estados



Describa claramente,

a) La función de cada estado.

b) La razón de cada transición.

10. Desarrolle los siguientes ejercicios bajo el sistema operativo GNU/Linux utilizando el lenguaje de programación C, el compilador GCC (GNU Compiler Collection) y el editor nano desde el intérprete de comandos.

a) Un proceso debe crear un proceso hijo y esperar por su finalización. El proceso hijo debe ejecutar un bucle en el cual se produzcan diez iteraciones, mostrando el número de iteración por la salida estándar. En particular, en cada iteración el proceso hijo debe suspenderse a sí mismo por un segundo.

b) Modifique el ejercicio anterior de forma tal que el proceso padre muestre por la salida estándar tanto su PID como el PID del proceso hijo. Análogamente, el proceso hijo debe mostrar su propio PID.

c) En base a su criterio, explique qué sucederá si el proceso padre finaliza *antes* de que lo haga el proceso hijo. Luego, implemente un programa que permita determinar si su respuesta era correcta.

d) Un proceso debe crear 10 procesos hijos. El primer proceso hijo debe suspenderse por 1 segundo, mientras que cada nuevo proceso creado debe suspenderse un segundo mas que el proceso previo. Finalmente, el padre debe esperar sincrónicamente en orden decreciente por la finalización de cada proceso hijo, utilizando para dicha tarea el ID otorgado por el kernel en cada paso de la creación. En otras palabras, el proceso padre deberá esperar en primera instancia por la finalización del último proceso hijo creado.

e) Ejecute y monitoree el estado de los procesos creados en el inciso anterior mediante el comando *ps*. ¿A qué conclusión puede llegar?

11. La secuencia de *Fibonacci* puede expresarse formalmente como:

$$fib_0 = 0$$
$$fib_1 = 1$$
$$fib_n = fib_{n-1} + fib_{n-2}$$

Siguiendo los lineamientos del ejercicio 10 escriba un programa que compute la secuencia de *Fibonacci* utilizando un nuevo proceso hijo. La cantidad de términos de la secuencia deber ser provisto como un parámetro al invocar el programa desde la línea de comandos. El proceso hijo debe mostrar por la salida estándar cada término computado.

12. Utilizando el intérprete de comandos bajo el sistema operativo GNU/Linux, ejecute las sentencias

```
$ cd path_to_programs
$ /bin/bash
$ ./programaX
```

donde *path_to_programs* representa la ruta hacia el directorio en el cual se encuentran almacenados los programas desarrollados en el ejercicio 10, mientras que *programaX* representa el nombre del programa implementado en el ejercicio $10a$. Muestre gráficamente las relaciones entre ***todos*** los procesos involucrados.

13. Determine el ámbito *(scope)* de una variable de entorno en el contexto del intérprete de comandos Bash. ¿Es posible que dos intérpretes de comandos compartan una variable? Si este fuera el caso, ¿corresponde efectivamente a un mecanismo IPC?

14. ¿Qué entiende por el término *Multithreading*?

15. Enuncie las características más importantes de los *threads* y especifique en qué casos los utilizaría.

16. ¿Cuál es la diferencia entre los *hilos a nivel de usuario (ULTs)* y *los hilos a nivel de kernel (KLTs)*? Describa ventajas y desventajas con respecto a la utilización de ULTs sobre KLTs.

17. Determine cúales de los siguientes componentes son compartidos entre los diversos hilos de ejecución de un proceso *multi-hilado*:

    a) Program Counter
    b) Registros de propósito general
    c) Instrucciones
    d) Stack Pointer
    e) Directorio de trabajo
    f) Variables globales
    g) Heap
    h) UID, GID

18. Considere un sistema multiprocesador y un programa multi-hilado escrito utilizando el modelo *muchos-a-muchos*. Suponga que el número de hilos a nivel de usuario en el programa es mayor que el número de procesadores en el sistema. Discuta las implicaciones en cuanto a rendimiento para los siguientes escenarios:

    a) El número de hilos a nivel de kernel asociados al programa es menor que el número de procesadores.

    b) El número de hilos a nivel de kernel asociados al programa es igual al número de procesadores.

    c) El número de hilos a nivel de kernel asociados al programa es mayor que el número de procesadores, pero es menor que el número de hilos a nivel de usuario.

19. ¿Cúal es la filosofía para el manejo de *threads* utilizada por el kernel del sistema operativo GNU/Linux? Determine cuál sería el resultado de la siguiente invocación:

```
clone(CLONE_VM | CLONE_FS | CLONE_FILES | CLONE_SIGHAND, 0);
```

20. ¿Qué entiende por POSIX y POSIX Threads (pthreads)?

21. Resuelva los siguientes ejercicios utilizando el API POSIX Threads:

    a) Un proceso debe crear n hilos de ejecución, donde n corresponde a un parámetro pasado por la línea de comandos. Cada hilo debe imprimir su identificador y finalizar. El proceso principal debe esperar por la finalización de todos sus hilos de ejecución.

    b) Un proceso debe crear n hilos de ejecución, donde n corresponde a un parámetro pasado por la línea de comandos. Cada hilo debe calcular la sumatoria de todos los números naturales que precedan a un argumento especificado. El primer argumento especificado corresponde a n, mientras que los restantes argumentos son mayores en una unidad a su predecesor. Es decir, el primer hilo recibirá como argumento n, mientras que el hilo k ($k \leq n$) recibirá como argumento $n + k$. Cada hilo debe imprimir por la salida estándar la sumatoria computada. El proceso principal debe esperar por la finalización de todos los hilos creados antes de finalizar.

    c) Repita el ejercicio anterior con la salvedad de que cada sumatoria computada sea mostrada por la salida estándar por el proceso principal, una vez que todos los hilos creados finalicen con su ejecución.

    d) Resuelva el ejercicio 11 de forma tal que el proceso principal utilice un nuevo hilo de ejecución para realizar el cálculo de la secuencia de Fibonacci. Una vez que la secuencia haya sido calculada, el proceso principal debe crear un nuevo hilo de ejecución cuyo objetivo sea mostrar por la salida estándar la secuencia calculada.

    e) Un dado proceso consta de m hilos de ejecución. Cada hilo de ejecución es denominado *worker* dada su inherente naturaleza para realizar trabajo útil por demanda. Cada *worker* puede realizar dos tareas especificas: calcular sumatorias o factoriales. Al crear un nuevo hilo de ejecución, el proceso principal comunica al mismo su identificador, un número entero y la operación que debe realizar. Al finalizar la ejecución de cada hilo, el proceso principal debe mostrar por la salida estándar el identificador del hilo, la operación realizada y el resultado obtenido.