ILI 246: Sistemas Operativos

Tarea #2

¡Arre mi pequeño pony!

Danilo Andrés Vergara Quitral Daniel Andrés Márquez Lutfy

8 de septiembre de 2014

1. Motivación

Cotidianamente nos encontramos con actividades que nos gustaría realizar simultáneamente, por ejemplo dormir y estudiar. Lamentablemente, muchas de éstas no son compatibles entre sí y generalmente optamos por realizarlas separadamente. De forma similar, los Sistemas Operativos se enfrentan a tareas que deben realizar simultáneamente para poder funcionar y satisfacer a sus usuarios, pero que, al igual que nosotros, no siempre son capaces de hacer al mismo tiempo, problema que se vuelve más difícil de resolver si el usuario añade más procesos. Por lo anterior, se han desarrollado diversos mecanismos, por ejemplo el *process scheduling*, que le permiten al SO lidiar con ésta problemática situación. Es por tanto objetivo de esta tarea ver cómo se comporta un programa que asigna varias tareas simultáneas, llamadas procesos, a nuestro SO.

2. La Tarea

La tarea está dividida en dos secciones. La primera es crear un programa escrito en C++ qué utilice las llamadas al sistema vistas en clases. La segunda consiste en un informe que debe incluir las respuestas que se harán con respecto a la primera parte.

2.1. Una carrera de ponys

Suponga que está viendo una carrera de ponys y se hace la siguiente pregunta ¿Cuál terminará primero?. Normalmente, es imposible saber cuál será el ganador, ya que existen muchos factores que afectan a cada pony: la dirección del viento mágico, la alineación de las estrellas, etc. Para recrear una situación similar, se le pide desarrollar un programa en C++ que realice lo siguiente:

1. Un proceso llamado **Juez de Partida** (el proceso padre) debe crear a 9 ponys (procesos hijos).
2. Los ponys deben esperar atentos la señal (un *signal*) de inicio de la carrera, la cual será realizada por el Juez de Partida tres segundos después de haber creado al último pony. Estos tres segundos son una cuenta regresiva que debe ser mostrada por pantalla.
3. Cada pony debe correr (ejecutarse), a lo menos, 10 segundos (usted le pide al pony correr por lo menos esa cantidad de tiempo). Al cumplirse este período de tiempo, el pony debe imprimir por pantalla "Pony #n ha terminado la carrera", donde #n es el número del pony según el orden en el que fue creado.
4. Cada pony debe entregar en un sobre (un *pipe*) al Juez de Partida la hora (hora del sistema operativo) en la que se retira de la carrera (se termina su ejecución).
5. Finalmente, el Juez de Partida debe imprimir el tiempo inicio de la carrera y de retirada de todos los ponys luego de irse el último de éstos.

2.1.1. Ejemplo de ejecución

```
[dmarquez@XWolf]$ make run
Iniciando Conteo Regresivo:
3
2
1
Pony 3 ha terminado la carrera
Pony 1 ha terminado la carrera
Pony 2 ha terminado la carrera
Pony 9 ha terminado la carrera
Pony 6 ha terminado la carrera
Pony 4 ha terminado la carrera
Pony 7 ha terminado la carrera
Pony 5 ha terminado la carrera
Pony 8 ha terminado la carrera
El juez de linea dice:
La carrera inicio a las 09:05:04
El pony 1 se retiro de la carrera a las 09:05:14
El pony 2 se retiro de la carrera a las 09:05:15
...
El pony 9 se retiro de la carrera a las 09:05:15
```

2.2. Informe

El informe debe ser realizado en LaTeXy debe incluir lo siguiente:

1. La información requerida como README en las reglas del curso.
2. Las repuestas a las siguientes preguntas y situaciones:
   - Ejecute su programa al menos cinco veces y observe el orden de impresión por pantalla. ¿Cambia el orden de impresión de cada pony? En caso de cambiar explique el porqué.
   - Repita el punto anterior abriendo y cerrando programas durante la ejecución de su programa ¿Influyen éstos en el resultado mostrado por el Juez? Justifique.
   - Explique la importancia de los tres segundos de espera antes de iniciar la carrera ¿soluciona completamente el problema que intenta evitar? Justifique.
   - Bono 5pts: Documente cualquier anormalidad que detecte durante la ejecución de su programa e intente explicar, lo mejor que pueda, la causa de ésta.

3. Restricciones y Consideraciones

- La tarea debe ser programada en **C++**, es libre de elegir la versión/estándar que más le acomode y que funcione en el labcomp.
- Para compilar y ejecutar el programa, se debe incluir un archivo **MakeFile**. Las instrucciones de cómo utilizarlo deben estar escritas en el informe.
- Se debe incluir el archivo .pdf y .tex de su informe **LaTeX** en la entrega.
- No incluya el archivo README para esta tarea.
- Su programa será revisado en los computadores del **Labcomp**, por lo que debe asegurarse de su correcto funcionamiento en ellos.
- Para la comunicación y creación de procesos, sólo se permite el uso de las llamadas al sistema presentadas en clases. Adicionalmente, no se permite el uso de *threads* y de ninguna biblioteca que facilite el uso, creación y gestión de procesos.
- No se permite utilizar las llamadas al sistema **exec** y/o **system**.
- Sólo se permite el uso de bibliotecas estándar de **C++** para realizar la tarea.
- Visite el siguiente **LINK** para obtener ayuda del cómo obtener la hora en su programa.

4. Entrega

- Debe subir a Moodle un archivo *gzip* con el nombre `numero_grupo-rol1-rol2.tar.gz` (por ejemplo `2-2010730204-28730225.tar.gz`), dentro del cual debe existir una carpeta con el mismo nombre y que debe contener, al menos, los siguientes archivos:
  - El PDF de su informe, junto al archivo .tex que lo genera.

  - El archivo *main* de su código fuente y el archivo MakeFile para su compilación y ejecución.
  - En el caso de que su programa se encuentre segmentado en varios archivos, se le permite nombrarlos e incluirlos a su criterio (siempre y cuando su programa funcione correctamente).

- La fecha de entrega será el **13 de Octubre** hasta las **00:05**.
- El incumplimiento de las reglas establecidas en el reglamento oficial, así como las restricciones mostradas en este documento, provocarán los descuentos pertinentes. Las copias serán sancionadas según el reglamento de tareas.