Sistemas Operativos
Tarea #2

Fabián Viani Gonzalez
201003048-2
Teodoro Hochfärber
201173039-9

October 13, 2014

README:

1. Supuestos: Se presuponen los elementos establecidos en la definicion de la tarea.
2. Archivos y Carpetas: Se entregan 4 archivos:
   (a) **main.cpp:** Este archivo contiene el codigo fuente correspondiente a la tarea.
   (b) **Makefile:** Este archivo contiene las intrucciones de compilacion. Contiene 4 targets: *tarea2*, el cual compila el codigo fuente, *clean*, el cual elimina los archivos generados por la compilacion, *mop* el cual elimina los archivos *.o dejados por la compilacion. Por ultimo tenemos *run* el cual corre el programa una vez compilado. Por default, se ejecutan los siguientes targets: *tarea2 mop run*, en ese orden.
   (c) **tarea2SO.tex:** Codigo fuente del presente informe. Debe ser compilado por herramientas LaTeX.
   (d) **tarea2SO.pdf:** Informe compilado y presentable.
3. Explicación de la Estrategia:
   En primer lugar, se establece un Signa Handler para permitir el manejo de señales. Creandolo antes de la creacion de nuevos procesos, nos aseguramos que todos tengan este Handler establecido. En segundo lugar, se generaron los procesos hijos nesesarios a partir de un loop que utiliza un fork. Esto, acompañado de un condicional, nos permite generar multiples procesos, todos descendencia directa del proceso principal. Una vez creados, se les deja en un loop infinito, a la espera de una señal desde el proceso padre. Cuando todos los procesos han sido creados, en el proceso padre se establece que señal se utilizara para llamar a los hijos. Se genera una llamada a todos los procesos que comparten PID de grupo con el proceso padre, incluyendo tambien una instruccion para que el proceso padre ignore la señal, mediante

```
signal(SIGINT, SIG_IGN);
kill(0, SIGINT);
```

   Justo antes de enviar la señal, se guarda el valor del tiempo presente, de forma de poder conocer el instante en el cual comienza la carrera.Una vez enviada la señal, cada proceso ejecuta el signal Handler establecido. Este los hace esperar 10 segundos, para despues guardar su tiempo de termino dentro de un pipe establecido en el proceso padre, avisando por pantalla el final de su ejecucion, para luego terminar esta. Mientras los procesos hijos hacen esto, el proceso padre espera a que terminen para poder continuar con su propia ejecución. Una vez sucede esto, el padre accede al pipe y lee los datos en este guardados. A partir de estos, muestra el tiempo en el cual cada proceso o "Pony" termino su carrera. Una vez echo esto, el programa termina.

Respuestas:

1. Si cambian en su orden de llegada, esto se debe a que el Sistema operativo no posee una priorización sincrónica para los procesos ni los threads. Estos son programas secuenciales en donde su velocidad es variable, solo el scheduler reconoce esta información.

2. El orden de los resultados mostrados por el Juez varía aún más. Esta vez es por un tema de recursos. En concurrencia, los distintos proceso e hilos en ejecución comparten los mismos recursos, que sumados a otros utilizan el tiempo de procesamiento. A la hora de abrir y cerrar programas, se están cambiando los estados de los distintos hilos y procesos (creación, asignación, bloqueo) que conllevan un cambio también en la asignación de recursos, por lo que el orden de llegada de cada "pony" llega a diferir más uno de otro.

3. La idea de realizar la espera de los 3 segundos, es que se alcance a realizar la creación de todos los procesos (ponys), para que de esta forma, el primer "pony" en crearse no inicie la carrera cuando el último pony aún no es creado. No podriamos decir a ciencia cierta que soluciona totalmente este problema, ya que no somos el scheduler y no conocemos la cantidad de tiempo de creación (no podemos saberlo con certeza, depende del equipo).