Sistemas Operativos

Departamento de Computación – FCEyN – UBA
Primer cuatrimestre de 2017

Nombre y apellido: _____

Nº orden: _____ L.U.: _____ Cant. hojas: _____

1	2	3	4	Nota

Recuperatorio del primer parcial – 27 de junio de 2017

ACLARACIONES: 1) **Numere** las hojas entregadas. Esta hoja se entrega y es la hoja cero. Complete en la primera hoja la cantidad total de hojas entregadas (sin contar el enunciado). 2) Realice cada ejercicio en **hojas separadas** y escriba **nombre, apellido y L.U. en cada una**. 3) Cada ejercicio se califica con **Bien**, **Regular** o **Mal**. La división de los ejercicios en incisos es meramente orientativa. Los ejercicios se califican globalmente. El parcial se aprueba con 2 ejercicios bien y a lo sumo 1 mal/incompleto. 4) El parcial **NO** es a libro abierto.

Justifique *adecuadamente* cada una de sus respuestas.

Ejercicio 1.

Responda verdadero o falso **justificando**:

a) Cualquier implementación de `fork()` debe crear otro proceso igual al que lo ejecuta y por ende se duplicar la memoria consumida en el sistema.

b) Si un proceso le pasa a otro a través de un pipe el puntero a una dirección de memoria el segundo proceso puede leer de esa dirección el valor que haya puesto el proceso padre.

c) Si un proceso le pasa a *un proceso hijo* a través de un pipe el puntero a una dirección de memoria el segundo proceso puede leer de esa dirección.

d) Para mejorar el rendimiento conviene lanzar dos procesos independientes en lugar de hacer `fork()` porque un proceso y todos sus hijos comparten el quantum.

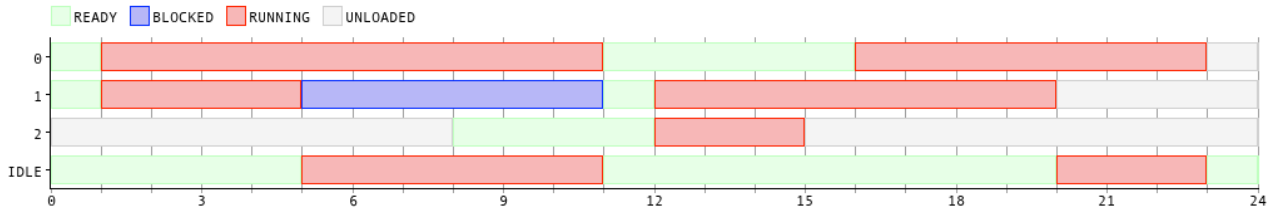
e) Se cuenta con dos operaciones que no realizan *syscalls* (A y B) y una tercera que sí (C). B debe ejecutarse sí o sí luego de C, mientras que entre A y B no hay dependencias. Sin embargo, por una cuestión de uso de cachés, es mejor que B se ejecute luego de A. Se proponen dos versiones del código: la primera es `A; C; B;`, la segunda es `C; A; B;`. Desde la perspectiva del aprovechamiento de los cachés, ambas son iguales.

Ejercicio 2.

Se tienen las siguientes tareas

```
TaskCPU 16
TaskAlterno 3 6 7
@8
TaskCPU 2
```

y un scheduler que genera la siguiente salida:



a) Calcular el *waiting time* y el *turnaround* promedios. Escriba el razonamiento de resolución de forma tal que pueda ser seguido por un ser humano (o un docente, que contrariamente a la opinión popular, es una subclase de humano).

b) Indique de qué tipo de *scheduler* se trata y cuáles son sus características. Indique claramente por qué llega a esa conclusión.

Ejercicio 3.

Se requiere implementar un *combinador* de texto. Se trata de un programa que ejecuta n programas que recibe por línea de comandos. A medida que van generando texto por salida estándar la lee en *round-robin*, de a 10 caracteres de cada uno y la va imprimiendo separada por `\n`.

Ejemplo:

```
prog1: Hola, estoy imprimiendo texto.
prog2: Yo también
prog3: Esto tampoco
prog1: Chau.
```

La salida de ejecutar `combinador prog1 prog2 prog3` debería ser

```
Hola, esto
Yo también
Esto tampo
Chau.
y imprimie
co
ndo texto.
```

Escriba el código del mencionado *combinador*.

Ejercicio 4.

Para cada una de los siguientes escenarios determine la combinación de estructura de datos y primitiva de sincronización más conveniente. **Jusitifique sus respuestas y escriba el fragmento de código que realiza la parte de sincronización en cada caso.**

a) Se tiene un productor y varios consumidores que deben consumir la información producida, que se genera esporádicamente. Cada pieza debe ser consumida por un consumidor distinto.

b) Ídem anterior pero la información se genera con altísima frecuencia y también se procesa rápidamente.

c) Ídem anterior pero se debe garantizar que los consumidores tienen acceso *round robin* a la información producida.

d) Se tienen varios productores que generan información numérica, y varios consumidores de esa información. Se cuenta además con una función `bool es_primo(int)` que se comporta como indica su nombre y toma un tiempo largo, proporcional al tamaño del parámetro que recibe (es decir, muy variable). Los productores generan la información muy velozmente. Los consumidores deben computar de manera colaborativa la cantidad total de números primos producidos. Alguien propone utilizar un spinlock para la sincronización entre productortes y consumidores y un semáforo para la sincronización entre consumidores para actualizar la cuenta de números primos. ¿Lo considera una solución correcta?