Fernando.

Sistemas Operativos

Departamento de Computación – FCEyN – UBA
Primer cuatrimestre de 2018

Nombre y apellido: _____

Nº orden: _____ L.U.: _____ Cant. hojas: 5

1	2	3	4	Nota
B	B	B	B	A

Primer parcial – 26/4 - Primer cuatrimestre de 2018

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. 5) **Justifique adecuadamente cada una de sus respuestas.**

Ejercicio 1.

Los programas en Unix como `ls` y `wc` están diseñados respetando una API para componerse de manera sencilla. Por ejemplo, se los puede componer fácilmente para contar la cantidad de archivos en un directorio con el siguiente comando de shell: `ls -l | wc -l`.

a) ¿Qué garantías debe cumplir el programador para que sus programas se puedan componer con otros de la misma manera?
b) ¿Qué herramientas provee el sistema operativo para que esta composición funcione?
c) Implemente el programa `superlswc` que cuenta la cantidad de archivos en una carpeta. Asuma que el sistema operativo ya cuenta con los programas `ls` y `wc`. No se permite utilizar la función **system**.

Ejercicio 2.

Un sistema monoproceso tiene un clock que llama cada 10ms a la función `sched` del kernel, la cual es la encargada de decidir si un proceso continúa usando el procesador o si debe ser reemplazado por otro proceso.

Para eso cuenta con las siguientes funciones auxiliares para comunicar su decisión:

- `void ponerACorrer(int pid)`: Realiza el cambio de contexto para poner a correr el proceso con el identificador pasado por parámetro.
- `void mantenerProceso()`: Mantiene corriendo el mismo proceso que estaba corriendo en la CPU.
- `bool procesoBloquedo(int pid)`: Indica si el proceso hizo una llamada de E/S y está listo para bloquearse.
- `void noHayNingunProcesoParaCorrer()`: Avisa que para el siguiente clock no existen procesos de usuario que puedan utilizar la CPU.

Además existen otras dos llamadas al kernel cuando ocurren eventos que son importantes para `sched`.

- Cada vez que un proceso ingresa al sistema y está disponible para correr (ready), se llama a la función `void ingresoProceso(int pid)`.
- Cada vez que un proceso que se había bloqueado por E\S está disponible para correr nuevamente (ready), se llama a la función `void volvioProceso(int pid)`

Implemente la función `sched` para que respete la política de Round Robin con un quantum de 50ms (recuerde que la funcione es llamada automáticamente cada 10ms). Deberá implementar las funciones `ingresoProceso` y `volvioProceso` para el correcto funcionamiento de `sched`.

Nota: Asuma que los procesos siempre necesitan consumir todo su quantum, y que si requieren hacer E\S dejarán todo preparado para bloquearse por E\S sólo al fin del ciclo del quantum donde hayan hecho una llamada de E\S.

Ejercicio 3.

Responda verdadero y falso. Justifique

a) La memoria virtual es útil para el uso de la glibc (biblioteca de C de GNU).
b) La memoria virtual es útil para la creación de nuevos procesos.
c) El bit de referenciado de una página en una tabla de páginas se mantiene por compatibilidad con sistemas anteriores.
d) Para el caso del acceso a las siguientes páginas: 1,2,3,4,1,2,5,1,2,3,4,5, el algoritmo de reemplazo de paginas LRU produce una anomalía de Belady si se aumenta de marcos de página de tres a cuatro.

Nota: La anomalía de Belady se produce cuando se aumenta la cantidad de marcos de página en un algoritmo de reemplazo pero al mismo tiempo aumenta la cantidad de fallos de página para una secuencia dada.

Ejercicio 4.

Se desea implementar un sistema para la Bolsa de Comercio que atienda llamados de clientes y realice el intercambio de acciones. Cada cliente cuenta con un único **Portfolio**, y por simpleza supondremos que existe un único tipo de acción.

El **Portfolio** de un cliente se obtiene mediante la función `Portfolio damePortfolio(int id)`. Cada **Portfolio** tiene una variable global que indica la cantidad de acciones que tiene el cliente y dos métodos `boolean vender(int num)` y `boolean comprar(int num)`, que disminuyen y aumentan la cantidad de acciones del clientes según el siguiente código:

```
void init() {
 cantidadAcciones = 0;
}

boolean vender(int num) {
 if (cantidadAcciones>=num && num>0) {
   cantidadAcciones -= num;
   return true;
 }
 return false;
}
```

```
boolean comprar(int num) {
 if (num > 0) {
   cantidadAcciones += num;
   return true;
 }
 return false;
}
```

Por otro lado, tenemos el siguiente proceso que se inicia en cuanto se prende el sistema de la Bolsa.

```
// Generador cuántico
while (true) {
 // Obtener un número entre 0 y 1 a partir del ruido cósmico
 double azar = RuidoCosmico();
 // Normalizarlo entre 0 y 999
 agenteSeleccionado = (azar * 1000) % 1000;
}
```

Por último, se cuenta con n procesos **Agentes** encargados de procesar el intercambio de acciones entre dos clientes. Cada **Agente** procesará un pedido distinto y para eso esperará mediante las funciones bloqueantes `int esperarVendedor()` e `int esperarComprador()` que se conecten dos clientes al otro lado de los respectivos socket y obtendrá con la función `int montoTransaction(int idComprador, int idVendedor)` el número de acciones de la transacción a realizar entre ambos. Una vez recibido el monto, procederá a confirmar la transacción en el sistema cuando `agenteSeleccionado` corresponda al número del agente.

```
Agente
while (true) {

        int idComprador = esperarComprador();
        int idVendedor = esperarVendedor();

        int monto = montoTransaction(int idComprador, int idVendedor)

        Portfolio portfolioComprador = damePortfolio(int idComprador);
        Portfolio portfolioVendedor = damePortfolio(int idVendedor);

        if (agenteSeleccionado == miNumeroDeAgente) {
                if (portfolioVendedor.vender(monto)) {
                        portfolioComprador.comprar(monto);
                }
        }
}
```

a) Asumiendo que `RuidoCosmico()` es una función que retorna un valor muy rápido, completar y/o modificar cuando sea necesario el código de los **Portfolios**, el **Generador Cuántico** y los **Agentes** para evitar problemas de sincronización, respetando el funcionamiento descripto. Argumente sobre la elección de cada mecanismo o modificación realizada.

b) Analice qué efectos puede causar en el sistema si RuidoCosmico() es una función que tarda mucho tiempo en retornar un resultado. ¿Cambiaría su solución propuesta? Justifique.

c) ¿Cumplen las dos variantes del sistema la propiedad de Ecuanimidad (*fairness*)? Justifique.