Sistemas Operativos - Primer parcial

25/4 - Primer cuatrimestre 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 **apellido, nombre 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) El programa `int main(int argc, char *argv[]) { printf("Hola"); return 0; }` no hace ninguna *syscall*.

b) Supongamos que se compilar el programa `int main(int argc, char *argv[]) { system("date"); return 0; }` generándose un archivo ejecutable. Supongamos también que a la hora de ejecutarlo, luego de cargar ese archivo a memoria, falla el disco por completo. Sin embargo, el programa podrá ejecutarse sin problemas porque ya fue cargado en memoria.

c) En cierto SO la función `gettimeofday()` que consulta el reloj de hardware del sistema requiere solamente 3 instrucciones para hacerlo, mediante la lectura de un registro. El programa A consiste en `for (i = 0; i<10000; i++) {i++;}` mientras que el programa B es `struct timeval hora; gettimeofday(&hora, NULL);`. El programa A va ejecutar más lento que el B porque tiene que ejecutar 10000 operaciones mientras que en el B sólo se llama a `gettimeofday()` que lee un registro del clock mediante 3 instrucciones.

d) Si un programa ejecuta `fork()` las variables definidas antes son compartidas mientras que las modificadas luego del fork() no se comparten. Sin embargo, esto puede solucionarse mediante el uso de punteros a la variable original.

Ejercicio 2.

Se cuenta con un sistema con 2 procesadores, y un scheduler FCFS sin desalojo por E/S, ni afinidad de procesador. El costo de cargar un proceso es de 2 s y el del cambio de contexto es de 1 s.

a) Realizar el diagrama de Gantt para los siguiente procesos:

PID	Llegada	T. ejec	T. bloq (inc.)
1	0	6	3-4
2	0	7	
3	1	3	
4	3	6	3-4

Donde el tiempo de bloqueo se contabiliza dentro del tiempo de ejecución y los tiempos de bloqueos están incluidos. Es decir, si una tarea tiene tiempo de ejecución de 5 segundos y bloquea en 2-3, ejecutará en $T = 0$, $T = 1$, bloqueará en $T = 2$, $T = 3$ y ejecutará en $T = 4$ (suponiendo que es la única tarea en el sistema). A su vez se contabilizará como que ejecuta 5 s.

b) Calcular el **waiting time** y **turnaround** promedio. Justificar mediante las cuentas que avalan el resultado.

c) ¿Cómo cambia el diagrama en caso de contarse con afinidad de procesador?

Ejercicio 3.

`tee` es un comando estándar del sistema operativo que reenvía lo que recibe por entrada estándar tanto a la salida estándar como a un archivo que recibe como parámetro, y sirve para, por ejemplo, guardar un log mientras se lo monitorea por pantalla.

Se requiere implementar el comando `teex`, una especie de `tee` extendido, que en lugar de un parámetro recibe tres, el primero correspondiente al programa a ejecutar, el segundo correspondiente a un archivo de salida, como en `tee` tradicional, y el tercer correspondiente a un archivo de entrada, de manera tal que el efecto de ejecutar

```
teex ./programa salida.log entrada.input
```

sea equivalente a

```
cat entrada.input | ./programa | tee salida.log
```

Ejercicio 4

En cierta ciudad se puso de moda un entretenimiento que consiste en entrar a una habitación con capacidad para 20 personas y escuchar 15 minutos de música generada al *pseudo azar*. Se dice que es pseudo azar porque el mecanismo que genera la música toma un número como semilla que es provista por alguno de los participantes (salvo la primera vez donde se inicializa en cero). La empresa que comercializa este entretenimiento lo hace en turnos sucesivos, para maximizar su ganancia. Sin embargo, un turno nuevo no puede entrar hasta que la sala no se haya vaciado.

Cuando llegan al evento, los participantes disponen de la función `Participante.sumarse_a_la_cola()`, así como de `Participante.irse_a_casa()` cuando deciden retirarse. La sala cuenta con un coordinador que va dando el ingreso mediante la función `Coordinador.permitir_ingreso_de_participante(unsigned int id_participante)`, que toma como parámetro el DNI con el que se identifican unívocamente los participantes.

La empresa tiene la política de seleccionar a dos de los participantes salientes para que propongan la nueva semilla. No los selecciona necesariamente por orden de salida, simplemente elige a dos cualesquiera, porque todos quieren ser los elegidos para encolar su semilla en la cola de dos posiciones que consulta el coordinador. El coordinador de la sala examina ambas y luego decide cuál de los dos toma mediante la función `Coordinador.elegir_favorita(int propuesta_semilla_1, int propuesta_semilla_2)` y se la pasa a la máquina ejecutando la función `Máquina.programar_semilla_pseudo_azar(int semilla)`. El participante seleccionado se gana la posibilidad de entrar de nuevo a la sala, junto con 19 otros participantes nuevos.

El código del coordinador es el siguiente:

```
int semilla = 0;
unsigned int lugares_disponibles = 20;
semáforo máquina_terminó = 0;
semáforo máquina_comienza = 0;
Cola<int, unsigned int> propuestas_semillas = new Cola<int, unsigned int>;

while (true) {
    // Dejo entrar a los 20 participantes.
    ... {
        ...
        id_participante = ...
        this.permitir_ingreso_de_participante(id_participante);
        ...
    } --> Completar este código considerando la variable lugares_disponibles.

    // Activo la máquina y espero que termine.
    máquina.programar_semilla_pseudo_azar(semilla);
    máquina_comienza.signal();
    máquina_terminó.wait();

    // Espero las dos propuestas de nueva semilla.
    ...
    <propuesta_semilla_1, id_participante_1> = propuestas_semillas.desencolar();
    <propuesta_semilla_2, id_participante_2> = propuestas_semillas.desencolar();

    // Elijo la que más me gusta.
    semilla = this.elegir_favorita(propuesta_semilla_1, propuesta_semilla_2);

    // Espero a que se vacíe la sala.
    ...

    // Dejo entrar al participante seleccionado e informo al otro.
    if (semilla == propuesta_semilla_1) {
         ...
    } else {
        ...
    }

    lugares_disponibles = 19;
}
```

a) Elegir estructuras de datos apropiadas para representar la fila de ingreso y la sala propiamente dicha.

b) Completar el código del coordinador (las líneas punteadas) y escribir el código de los procesos que representan a los participantes, resolviendo apropiadamente los desafíos de sincronización que surgen. Para eso se requiere definir los mútexes, semáforos y operaciones atómicas necesarias, junto con su inicialización y aclarar cuáles son atómicos y cuáles no.

c) Argumente por qué su programa es correcto y todo el mundo que hace la cola en algún momento entra.

d) ¿Podría producirse inanición en algún momento?