Posibles soluciones a los ejercicios del parcial práctico MC del 11

Importante: las soluciones que se muestran a continuación no son las únicas que se pueden considerar dos ejercicios planteados.

1. Resolver con *SEMÁFOROS* el siguiente problema. En una planta verificadora de vehículos, existen *7 estaciones* donde se dirigen 150 vehículos para ser verificados. Cuando un vehículo llega a la planta, el *coordinador* de la planta le indica a qué estación debe dirigirse. El coordinador selecciona la estación que tenga menos vehículos asignados en ese momento. Una vez que el vehículo sabe qué estación le fue asignada, se dirige a la misma y espera a que lo llamen para verificar. Luego de la revisión, la estación le entrega un comprobante que indica si pasó la revisión o no. Más allá del resultado, el vehículo se retira de la planta. *Nota:* maximizar la concurrencia.

```
int cant_estaciones[7] = {0} [7];    int estaciones_asignadas[150];
sem sem_cont_estaciones = 1, sem_entrada = 1, sem_atencion_entrada = 0;
sem sem_espera_estacion[150] = {0} [150], sem_atencion_verificacion[7] = {0} [7];
sem sem_estacion[7] = {1} [7],  sem_espera_resultado[150] = {0} [150];

string resultado [150];
queue atencion_entrada, estacion[7];

Process Vehículo [i::1..150] {

    // encola su ID para que lo atienda el coordinador
    P(sem_entrada);
    push(atencion_entrada, i);
    V(sem_entrada);
    // avisa para que lo atienda el coordinador
    V(sem_atencion_entrada);
    // espera estacion
    P(sem_espera_estacion[i]);
    // copia
    mi_estacion = estaciones_asignadas[i];
    // encola su ID para que lo atiendan la estación
    P(sem_estacion[mi_estacion]);
    push(estacion[mi_estacion],i);
    V(sem_estacion[mi_estacion]);
    // avisa para que lo atiendan en la estación
    V(sem_atencion_verificacion[mi_estacion]);
    // espera resultado
    P(sem_espera_resultado[i]);
    // decrementa contador para mantener valor actualizado
    P(sem_cont_estaciones);
    cant_estaciones[mi_estacion]--;
    v(sem_cont_estaciones);
}
```

```
Process Entrada {
    int id_min_estacion, id;

    while (true) {
        // espera pedido de atencion
        P(sem_atencion_entrada);
        // desencola pedido con exclusión mutua
        P(sem_entrada);
        id = pop (atencion_entrada);
        V(sem_entrada);
        // busca el mínimo con exclusión mutua
        P(sem_cont_estaciones);
        id_min_estacion = min(cant_estaciones); // retorna la posición de la cantidad mínima
        cant_estaciones[id_min_estacion]++; // incrementa para actualizar celda
        V(sem_cont_estaciones);
        estaciones_asignadas[id] = id_min_estacion; // asigna estación
        V(sem_espera_estacion[id]); // le avisa para que pueda continuar

    }
}

Process Estacion[i::1..7] {
    int id;
    while (true) {
        // se bloquea a la espera de que haya vehiculos
        P(sem_atencion_verificacion[i]);
        // desencola vehiculo
        P(sem_estacion[i]);
        id = pop(estacion[i]);
        V(sem_estacion[i]);
        // verificar
        resultado[id] = verificar(id);
        // le avisa que ya está el resultado disponible
        V(sem_espera_resultado[id]);
    }
}
```

2. Resolver con MONITORES el siguiente problema. En un sistema operativo se ejecutan *20 procesos* que periódicamente realizan cierto cómputo mediante la función *Procesar()*. Los resultados de dicha función son persistidos en un archivo, para lo que se requiere de acceso al subsistema de E/S. Sólo un proceso a la vez puede hacer uso del subsistema de E/S, y el acceso al mismo se define por la prioridad del proceso (menor valor indica mayor prioridad).

```
Monitor SubsistemaES {
    int esperando = 0;
    int usando = 0;
    cond colas[N];
    Queue(int,int) en_espera;

    procedure pedir (int id, int prioridad) {
        if (usando > 0){ // está ocupado
            esperando++; // incrementar contador para indicar que hay uno más en espera
            push(en_espera, (prioridad, id)); // inserta ordenado por prioridad
            wait(colas[id]); // dormir en cola condition individual
        }
        else // está libre
            usando++; // marcar como ocupado
    }

    procedure liberar() {
        if (esperando > 0){ // si hay procesos esperando
            int id = pop(en_espera); // seleccionar el de mayor prioridad
            signal(colas[id]); // despertar al de mayor prioridad
            esperando--; // decrementar para indicar que hay uno menos en espera
        } else
            usando--;   // marcar como libre
    }
}


Process Proceso [i: 1..20]{
    int prioridad = obtenerPrioridad();
    While (true) {
        // computar
        resultados = Procesar();
        // solicitar acceso
        SubsistemaES.pedir(i,edad);
        // usar subsistema E/S
        Persistir(resultados);
        // liberar
        SubsistemaES.liberar();
    }
}
```