RESUMEN PROGRAMACIÓN CONCURRENTE – 2DO PARCIAL

PASAJE DE MENSAJES SINCRÓNICOS:

Pedidos con Admin, con orden de llegada. Un solo "Empleado"

```
a) Analice el problema y defina qué procesos, recursos y comunicaciones serán
necesarios/convenientes para resolverlo.



Process Examinador [id: 0..R-1]{
    SitioWeb sitio;
    while (true){
        buscarVirus(sitio);
        Admin!reporte(sitio);
    }
}

Process Analizador{
    SitioWeb sitio;
    whilev(true){
        Examinador!pedido();
        Examinador?reporte(sitio);
        resolverVirus(sitio);
    }

}

Process Admin {
    cola buffer;
    SitioWeb s;

    while (true){
        do Examinador[*]?reporte (sitio)  --> buffer.push (s);

        not empty (buffer); Analizador?pedido() --> Analizador!reporte(pop(buffer));
    }
}
```

Empleados con distintas funciones

```
Process Empleado1{
    text muestra;
    while(true){
        muestra := prepararMuestra();
        Admin!.envio(muestra);
    }
}

Process Empleado2{
    text muestra;
    while(true){
        Admin!pedido();
        Admin?trabajo(muestra);
        armarSet();
        Empleado3!enviarMuestra(muestra);
        Empleado3?analisis();
    }
}

Process Empleado3{
    text muestra;
    do Empleado2?enviarMuestra (muestra) --> Empleado2!analisis (devolverAnalisis());

}

Process Admin{
    cola buffer;
    text muestra;

    do Empleado1?envio (muestra)  --> buffer.push (muestra);

    not empty (buffer); Empleado2?pedido() --> Empleado2!trabajo(pop(buffer));
}
```

P profesores para corregir un examen. Deben terminar su ejecución.

```
b) Considerando que P>1.

Process Alumno[id:0..N-1]{
    int miNota;
    hacerExamen();
    Admin!termine(examen,id);
    Profesor?devolverNota(miNota);
}

Process Profesor[id:0..P-1]{
    int idAlumno, nota; text examen;
    bool ok := true;
    while (ok){
       Admin!pedido(id);
       Admin?trabajo(examen,idAlumno);
       if (idAlumno <> -1){
        corregirExamen(idAlumno,nota);
        Alumno[idAlumno]!.devolverNota(nota);
       }
       else{
          ok := false;
       }
    }
}
```

```
Process Admin{
    cola examenes;
    text examen;
    int idAlumno, idProfesor;
    int P,N;

    for (i: 0..N*2-1){
        do Alumno[*]?termine (examen,idAlumno)  --> examenes.push (examen,idAlumno);

        not empty (examenes); Profesor[*]?pedido(idProfesor) --> Profesor[idProfesor]!trabajo(examenes.pop(examen,id));
    }

    for (i: 0..P-1){
        Profesor[i]!trabajo(-1);
    }

}
```

Idem anterior, pero con barrera cuando llegan los alumnos.

```
c) Idem b) pero considerando que los alumnos no comienzan a realizar su examen hasta
que todos hayan llegado al aula.

Process Alumno[id:0..N-1]{
    int miNota;
    Admin!llegue();
    Admin?empezar();
    hacerExamen();
    Admin!termine(examen,id);
    Profesor?devolverNota(miNota);
}

Process Profesor[id:0..P-1]{
    int idAlumno, nota; text examen;
    bool ok := true;
    while (ok){
       Admin!pedido(id);
       Admin?trabajo(examen,idAlumno);
       if (idAlumno <> -1){
        corregirExamen(idAlumno,nota);
        Alumno[idAlumno]!.devolverNota(nota);
       }
       else{
          ok := false;
       }
    }
}
```

```
Process Admin{
    cola examenes;
    text examen;
    int llegada = 0;
    int idAlumno, idProfesor;

    for (i: 0..N-1){
        Alumno[*]?llegue();
    }

    for (i: 0..N-1){
        Alumno[i]!empezar();
    }

    for (i: 0..N*2-1){
        do Alumno[*]?termine (examen,idAlumno)  --> examenes.push (examen,idAlumno);

        not empty (examenes); Profesor[*]?pedido(idProfesor) --> Profesor[idProfesor]!trabajo(examenes.pop(examen,id));
    }

    for (i: 0..P-1){
        Profesor[i]!trabajo(null,-1);
    }
}
```

Exclusión mutua

```
a) Implemente una solución donde el empleado sólo se ocupa de garantizar la exclusión
mutua (sin importar el orden).


Process Persona [id: 0..P-1]{
    Empleado!.llegue(id);
    Empleado?empezar();
    usarSimulador();
    Empleado!.listo();
}

Process Empleado{
    int idP;
    while(true){
        Persona?llegue(idP);
        Persona[idP]!empezar();
        Persona?listo();
    }
}
```

Idem anterior pero por orden de ID

```
b) Modifique la solución anterior para que el empleado los deje acceder según el orden de
su identificador (hasta que la persona i no lo haya usado, la persona i+1 debe esperar).


Process Persona [id: 0..P-1]{
    Empleado!.llegue();
    Empleado?empezar();
    usarSimulador();
    Empleado!.listo();
}

Process Empleado{
    int sig = 0;
    while(true){
        Persona[sig]?llegue();
        Persona[sig]!empezar();
        Persona[sig]listo();
        sig++;
    }
}
```

Passing the Condition

```
5. En un estadio de fútbol hay una máquina expendedora de gaseosas que debe ser usada por
E Espectadores de acuerdo con el orden de llegada. Cuando el espectador accede a la
máquina en su turno usa la máquina y luego se retira para dejar al siguiente.
Nota: cada Espectador una sólo una vez la máquina.

Process Espectador [id: 0..E-1]{
    Admin!llegue(id);
    Admin?empezar();
    // usar maquina
    Admin!listo();
}

Process Admin{
    int idP; bool libre := true; cola llegada;
    do Espectador[*].llegue? -->
            if (libre){
                libre := false;
                Espectador[idP]!empezar();
            }
            else{
                llegada.push(idP);
            }
    [] Espectador[*]?listo -->
            if (not empty llegada){
                Espectador[llegada.pop()]!.empezar;
            }
            else{
                libre := true;
            }
}
```

Más de un recurso, utilizando DO.

```
1) En una oficina existen 100 empleados que envían documentos para imprimir en 5 impresoras
compartidas. Los pedidos de impresión son procesados por orden de llegada y se asignan a la primera
impresora que se encuentre libre:

b) Resuelva el problema anterior usando PMS.

Process Empleado [id: 1..100]{
    texto Doc;
    Doc := generarDocumento ();
    Admin!Pedido (idC,Doc);
    Impresora[*]?Respuesta(Doc);
}

Process Impresora [id: 1..5]{
    int idE; texto d;
    while (true){
        Admin!Trabajo(id);
        Admin?Siguiente(idE,Doc);
        imprimirDocumento (Doc);
        Empleado[idE]!Respuesta(Doc);
    }
}

Process Admin {
    int idI,idE; texto d; cola buffer;
    do Empleado[*]?Pedido(idE,d) ---> buffer.push(idE,d);
        not empty (buffer); Impresora[*]?Trabajo(idI) --->  d := buffer.pop(); idE := buffer.pop();
                                                            Impresora[idI]!Siguiente(idE,d);

    od
}
```