Programación Concurrente ATIC/Redictado Programación Concurrente - Primera fecha de MD - 18/06/2025 - Tema 1

1. Resolver con **PASAJE DE MENSAJES ASINCRÓNICOS (PMA)** el siguiente problema. Hay ***E estudiantes*** que rinden un examen final y ***un profesor*** que cuando todos los estudiantes han llegado les entrega el enunciado. Luego el profesor va corrigiendo los exámenes y enviando la nota de acuerdo con orden en que se van entregando. Cada alumno realiza el examen, lo entrega y espera a que el profesor le indique la nota. ***Nota:*** maximizar la concurrencia; no realizar *busy waiting*; todos los procesos deben terminar.

2. Resolver con **PASAJE DE MENSAJES SINCRÓNICOS (PMS)** el siguiente problema. En un consultorio hay ***un médico*** para atender a ***15 pacientes*** de acuerdo con el orden de llegada. Cada paciente al llegar espera a que el médico lo atiende y le indique su tratamiento. ***Nota:*** existe la función *atender()* que simula que el médico está atendiendo a un paciente; todos los procesos deben terminar.

3. Resolver con **ADA** el siguiente problema. En una competencia de programadores hay ***P participantes*** y ***UN coordinador***. El coordinador entrega el problema a resolver a los participantes y recibe las resoluciones para corregir. Cada participante debe conocer el resultado de su trabajo y el orden en que entregó su resolución. ***Nota:*** maximizar la concurrencia.

Pedro Carballo 14350/7

1) PMA

A

1	B
2	B
3	B

Pedro Carballo
14350/7


```
chan llegada();
chan enunciado(txt);
chan entrega(txt, id);
chan resul[E](double);
```

// se asume que el profesor ya posee el enunciado
// se asume que es el mismo enunciado para todos los estudiantes.

```
process Estudiante(id:1..E)
{ txt enun;          double nota;
  txt resolucion;
  //llega
  Send llegada();              // avisa que llega
  receive enunciado(enun);     //espera enunciado
  resolucion = resolveExamen(enun);  //resuelve examen
  send entrega(resolucion, id);      // entrega examen
  receive resul[id](nota);           //espera nota
  //se va
}
```

```
process Profesor
{ double nota;      int idE;
  txt enun; // ya tiene el enunciado
  txt resolucion;

  For i=1 to E
  { receive llegada();  // espera llegada estudiantes
  }
  for i=1 to E
  { send enunciado(enun); // entrega enunciados
  }
  for i=1 to E
  { receive entrega(resolucion, idE);
    //espera resolucion de alumno →
    // corrige examen alumno →
    nota = corrigeExamen(resolucion);
    // entrega nota alumno →
    send resul[idE](nota);
  }
}
```

2) PMS

```
process Paciente (id: 1..15)
{ txt tratamiento;

  Admin ! Llegada (id);      // avisa que llega
  Medico ? Atencion ();      // espera atención
  // es atendido
  Medico ? Rta (tratamiento); // espera tratamiento y se va
}
```

```
process Medico
{ txt tratamiento;
  int idP;

  for i=1 to 15
  { Admin ! Solicitud(),     // avisa que está listo
    Admin ? Siguiente (idP); // espera siguiente
    Paciente [idP] ! Atencion (); // avisa paciente
    // atender ();
    // tratamiento = decideTratamiento();
    Paciente [idP] ! Rta (tratamiento); // indica tratamiento
  }
}
```

```
process Admin
{ cola buffer;   int idP;

  for i=1 to 30
  { if  Paciente[*] ? Llegada(idP); ⇒ push(buffer (idP));

    □ (not empty (buffer)); Medico ? Solicitud (); ⇒ Medico ! Siguiente (pop (buffer (idP)));

  }
}
```

Pedro Carballo
14350/7

3) ADA

```
//Se asume que se
entrega el mismo problema
a todos los participantes

PROCEDURE Competencia IS                    //se asume que el coordinador ya
                                              pasó el problema
   TASK TYPE Participante;
   TASK Coordinador IS                      //no sé si el resultado es
                                              solo una nota o una corrección
                                              (x ahí es de tipo txt)
      ENTRY PidoProblema(p: OUT txt);
      ENTRY Entrega (resolucion: IN txt; resultado: OUT txt; orden: OUT int);
   END Coordinador;

   arrParticipantes: array (1..P) OF Participante;


   TASK BODY Participante IS
    p: txt;
    resolucion: txt;
    resultado: txt;
    orden: int;
   BEGIN
      Coordinador.PidoProblema(p);          //pide el problema
      resolucion = resuelveProblema(p);     //resuelve
      Coordinador.Entrega(resolucion, resultado, orden);   //entrega y recibe
                                                             corrección y num de orden
   END Participante;
```

```
TASK BODY Coordinador IS
 orden: int;

 Problema: txt = ...; // ya lo posee.

BEGIN
   orden=0;
   FOR i IN 1..2*P LOOP
        SELECT
            ACCEPT PidoProblema (p:OUT txt) DO
                  p= problema;
            END PidoProblema;

        OR
            ACCEPT Entrega (resolucion: IN txt; resultado: OUT txt; o: OUT int) DO
                  orden = orden +1;
                  o= orden;
                  resultado = corrige (resolucion);
            END Entrega;
        END SELECT;
   END Loop;
END Coordinador;


BEGIN
   null;
END Competencia;
```

// recibe pedido y entrega problema

// recibe resolución, corrige y entrega resultado y nº orden