Programación concurrente

Práctica 3 - Monitores

1)

a) El código funciona correctamente.

b)

```
1) b)

Monitor Puente {
	Procedure cruzar(){
		cruzar_puente();
	}
}

Process Auto [a = 1..M] {
	Puente.cruzar();
}
```

c) La solución original no respeta el orden de llegada.
d) La solución reescrita en el punto 1)b) tampoco respeta el orden de llegada.

2)

a) Se va a necesitar un proceso para los procesos que quieren leer, un monitor para el acceso a la base de datos, el cual limite la cantidad de consultas simultaneas.

b)

```
2) b)

Monitor MotorBD {
	int cant_consultas = 0;
	cond hay_espacio;

	Procedure solicitarAcceso() {
```

```
            while(cant_consultas == 5) {
                  wait(hay_espacio);
            }
            cant_consultas ++;
      }

      Procedure liberar() {
            cant_consultas --;
            signal(hay_espacio);
      }
}

Process Proceso [p = 1..N] {
      MotorBD.solicitarAcceso();
      usarBD();
      MotorBD.liberar();
}
```

3)

a)

```
3) a)

Monitor Fotocopiadora {
      Procedure usar() {
            Fotocopiar();
      }
}

Process Persona [p = 1..N] {
      Fotocopiadora.usar();
```

```
}
```

b)

```
3) b)

Monitor Fotocopiadora {
	libre = true;
	int cant_esperando = 0;
	cond puede_usar;

	Procedure solicitarAcceso() {
		if(!libre) {
			cant_esperando ++;
			wait(puede_usar);
			cant_esperando --;
		} else {
			libre = false;
		}
	}

	Procedure liberar() {
		if(cant_esperando > 0) {
			signal(puede_usar);
		} else {
			libre = true;
		}
	}
}

Process Persona [p = 1..N] {
	Fotocopiadora.solicitarAcceso();
```

```
      Fotocopiar();
      Fotocopiadora.liberar();
}
```

c)

```
3) c)

Monitor Fotocopiadora {
      Cola cola;
      bool libre = true;
      cond puedo_usar[N];

      Procedure solicitarAcceso(id: in int, edad: in int) {
            if(!libre){
                  cola.insertar_ordenado(id, edad);
                  wait(puedo_usar[id]);
            }
      }

      Procedure liberar() {
            int id;
            if(!cola.empty()) {
                  cola.dequeue(id);
                  signal(puedo_usar[id]);
            } else {
                  libre = true;
            }
      }
}

Process Persona [p = 1..N] {
```

```
      Fotocopiadora.solicitarAcceso(p, get_edad(p));
      Fotocopiar();
      Fotocopiadora.liberar();
}
```

d)

```
3) d)

Monitor Fotocopiadora {
      int id_a_usar = 0;
      cond puedo_usar[N];

      Procedure solicitarAcceso(id: in int) {
            if(id_a_usar != id) {
                  wait(puedo_usar[id]);
            }
      }

      Procedure liberar() {
            id_a_usar ++;
            signal(puedo_usar[id_a_usar]);
      }
}

Process Persona [id = 0..N-1] {
      Fotocopiadora.solicitarAcceso(id);
      Fotocopiar();
      Fotocopiadora.liberar();
}
```

e)

```
3) e)

Monitor Fotocopiadora {
     int cant_esperando = 0;
     bool libre = true;
     cond puede_usar, avisa_empleado, fotocopiadora_libre;

     Procedure solicitarAcceso() {
          signal(avisa_empleado);
          cant_esperando ++;
          wait(puede_usar);
     }

     Procedure dar_acceso() {
          if(cant_esperando == 0) {
               wait(avisa_empleado);
          }
          if(!libre) {
               wait(fotocopiadora_libre);
          } else {
               libre = false;
          }
          cant_esperando --;
          signal(puede_usar);
     }

     Procedure liberar() {
          if(cant_esperando == 0) {
               libre = true;
          } else {
               signal(fotocopiadora_libre);
```

```
            }
      }
}

Process Persona [p = 1..N] {
      Fotocopiadora.solicitarAcceso();
      Fotocopiar();
      Fotocopiadora.liberar();
}

Process Empleado {
      while(true) {
            Fotocopiadora.dar_acceso();
      }
}
```

f)

```
3) f) - Versión 1

Monitor Fotocopiadora {
      int fotocopiadora_a_usar[N];
      Cola fotocopiadoras_libres, cola;
      cond puede_usar, avisa_empleado, fotocopiadora_libre;

      Procedure solicitarAcceso(id: in int, nro_fotocopiadora: out
int) {
            signal(avisa_empleado);
            cola.enqueue(id);
            wait(puede_usar);
            nro_fotocopiadora = fotocopiadora_a_usar[id];
      }
```

```
      Procedure dar_acceso() {
            int id;
            if(cola.empty()) {
                  wait(avisa_empleado);
            }
            if(fotocopiadoras_libres.empty()) {
                  wait(fotocopiadora_libre);
            }
            cola.dequeue(id);

fotocopiadoras_libres.dequeue(fotocopiadora_a_usar[id]);
            signal(puede_usar);
      }

      Procedure liberar(nro_fotocopiadora: in int) {
            if(!cola.empty()) {
                  signal(fotocopiadora_libre);
            }
            fotocopiadoras_libres.enqueue(nro_fotocopiadora);
      }

      {
            for [i = 0..9] {
                  fotocopiadoras_libres.enqueue(i);
            }
      }
}

Process Persona [p = 0..N-1] {
      int nro_fotocopiadora;
```

```
      Fotocopiadora.solicitarAcceso(p, nro_fotocopiadora);
      Fotocopiar(nro_fotocopiadora);
      Fotocopiadora.liberar(nro_fotocopiadora);
}

Process Empleado {
      while(true) {
            Fotocopiadora.dar_acceso();
      }
}
```

f)

```
3) f) - Versión 2

Monitor Fotocopiadora [id = 0..9] {
      Procedure usar () {
            Fotocopiar();
      }
}

Monitor AccesoAFotocopiadora {
      bool fotocopiadoras_libres[10] = ([10] true);
      int fotocopiadora_a_usar, cant_personas = 0;
      cond fotocopiadora_libre, quiere_fotocopiar,
fotocopiadora_asignada;

      Procedure solicitar_acceso (nro_fotocopiadora: out int) {
            cant_personas ++;
            signal(quiere_fotocopiar);
            wait(fotocopiadora_asignada);
```

```
            nro_fotocopiadora = fotocopiadora_a_usar;
      }

      Procedure dar_acceso () {
            if(cant_personas == 0) {
                  wait(quiere_fotocopiar);
            }
            cant_personas --;

            # alguna_es_true retorna el indice de la primera
posicion true, o -1
            fotocopiadoras_libres.alguna_es_true(nro_fotocopiadora)
            if(nro_fotocopiadora == -1) {
                  wait(fotocopiadora_libre);

fotocopiadoras_libres.alguna_es_true(nro_fotocopiadora)
            }

            fotocopiadoras_libres[nro_fotocopiadora] = false;
            fotocopiadora_a_usar = nro_fotocopiadora;
            signal(fotocopiadora_asignada);
      }

      Procedure liberar (nro_fotocopiadora) {
            fotocopiadoras_libres[nro_fotocopiadora] = true;
            signal(fotocopiadora_libre);
      }
}


Process Persona [p = 0..N-1] {
```

```
      int nro_fotocopiadora;
      Fotocopiadora.solicitar_acceso(p, nro_fotocopiadora);
      Fotocopiadora[nro_fotocopiadora].usar();
      Fotocopiadora.liberar(nro_fotocopiadora);
}

Process Empleado {
      while(true) {
            Fotocopiadora.dar_acceso();
      }
}
```

4)

```
4)

Monitor Puente {
      Cola cola;
      int peso_total = 0;
      cond hay_espacio;

      Procedure solicitarAcceso(peso: in int) {
            if(!cola.empty() || peso_total + peso > 50_000) {
                  cola.enqueue(peso);
                  wait(hay_espacio);
                  cola.dequeue();
            }
            peso_total += peso;
      }
```

```
      Procedure liberar(peso: in int) {
            peso_total -= peso;

            int peso_prox_vehiculo;
            cola.peek(peso_prox_vehiculo);

            if(!cola.empty() && peso_total + peso_prox_vehiculo <=
50_000){
                  signal(hay_espacio);
            }
      }
}

Process Vehiculo [id = 0..N-1] {
      int peso = get_peso(id);
      Puente.solicitarAcceso(peso);
      pasar();
      Puente.liberar(peso);
}
```

5)

a)

```
5) a)

Monitor Corralon {
      bool llego_cliente = false;
      cond hay_cliente, empleado_libre, lista_entregada,
comprobante_listo;
      string comprobante[N];
      ListaProductos lista;
```

```
     int idCliente;

     Procedure entregarLista(lista_cliente: in ListaProductos,
compr: out string, id: in int) {
          llego_cliente = true;
          signal(hay_cliente);
          wait(empleado_libre);

          # El cliente es llamado para ser atendido
          lista = lista_cliente;
          signal(lista_entregada);
          int idCliente = id;
          wait(comprobante_listo);
          compr = comprobante[id];
     }

     Procedure obtenerLista(lista_cliente: out string) {
          if(!llego_cliente) {
               wait(hay_cliente);
          }
          llego_cliente = false;
          signal(empleado_libre);
          wait(lista_entregada);
          lista_cliente = lista;
     }

     Procedure entregarComprobante(compr: in string) {
          comprobante[idCliente] = compr;
          signal(comprobante_listo);
     }
}
```

```
Process Cliente [id = 0..N-1] {
      string comprobante;

      ListaProductos lista = getListaProductos();
      Corralon.entregarLista(lista, comprobante, id);
}

Process Empleado {
      string comprobante;

      for [_ 0..N-1] {
            Corralon.obtenerLista(lista);
            prepararComprobante(lista, comprobante);
            Corralon.entregarComprobante(comprobante);
      }
}
```

b)

```
5) b)

Monitor Corralon {
      int cant_clientes_esperando = 0;
      cond hay_cliente, empleado_libre, lista_entregada,
comprobante_listo;
      string comprobante[N];
      Cola listas;


      Procedure entregarLista(lista_cliente: in ListaProductos,
compr: out string, id: in int) {
```

```
            cant_clientes_esperando ++;
            signal(hay_cliente);
            wait(empleado_libre);

            # El cliente es llamado para ser atendido
            listas.enqueue(lista_cliente, id);
            signal(lista_entregada);
            wait(comprobante_listo);
            compr = comprobante[id];
      }

      Procedure obtenerLista(lista_cliente: out string, id: out
int) {
            while(cant_clientes_esperando == 0) {
                  wait(hay_cliente);
            }
            cant_clientes_esperando --;
            signal(empleado_libre);

            wait(lista_entregada);
            listas.dequeue(lista_cliente, id);
      }

      Procedure entregarComprobante(compr: in string, id: in int) {
            comprobante[id] = compr;
            signal(comprobante_listo);
      }
}

Process Cliente [id = 0..N-1] {
      string comprobante;
```

```
      ListaProductos lista = getListaProductos();
      Corralon.entregarLista(lista, comprobante, id);
}

Process Empleado [id = 0..E-1] {
      string comprobante;
      int idCliente;

      while (true) {
            Corralon.obtenerLista(lista, idCliente);
            prepararComprobante(lista, comprobante);
            Corralon.entregarComprobante(comprobante, idCliente);
      }
}
```

c)

```
5) c)

Monitor Corralon {
      ...
      int cant_clientes_atendidos = 0;



      Procedure entregarLista(lista_cliente: in ListaProductos,
compr: out string, id: in int) ...

      Procedure obtenerLista(lista_cliente: out string, id: out
int, todos_los_clientes_atendidos: out bool) {
            if(cant_clientes_atendidos < N){
                  ...
```

```
            } else {
                  todos_los_clientes_atendidos = true;
            }
      }

      Procedure entregarComprobante(compr: in string, id: in int) {
            comprobante[id] = compr;
            signal(comprobante_listo);
            cant_clientes_atendidos ++;
      }
}

Process Cliente [id = 0..N-1] ...

Process Empleado [id = 0..E-1] {
      string comprobante;
      int idCliente;
      bool quedan_clientes = true;

      while (true && quedan_clientes) {
            Corralon.obtenerLista(lista, idCliente,
quedan_clientes);
            prepararComprobante(lista, comprobante);
            Corralon.entregarComprobante(comprobante, idCliente);
      }
}
```

6)

```
6)

Monitor Comision {
	int cant_alumnos_que_llegaron = 0, nro_grupo[50],
cant_tareas_completadas = 0, notas[25];
	cond llegaron_todos, nro_grupo_asignado[50],
tarea_completada, nota_lista[25];
	Cola tareas_completadas;

	Procedure hacerFila(id: in int, nro_grupo_alumno: out int){
		cant_alumnos_que_llegaron ++;
		if(cant_alumnos_que_llegaron == 50) {
			signal(llegaron_todos);
		}

		wait(nro_grupo_asignado[id]);
		nro_grupo_alumno = nro_grupo[id];
	}

	Procedure avisarFinTarea(nro_grupo_alumno: in int, nota: out
int) {
		cant_tareas_completadas ++;
		signal(tarea_completada);
		tareas_completadas.enqueue(nro_grupo);

		wait(nota_lista[nro_grupo_alumno]);
		nota = notas[nro_grupo_alumno];
	}

	Procedure esperarATodos(){
```

```
          if(cant_alumnos_que_llegaron < 50) {
               wait(llegaron_todos);
          }
     }

     Procedure asignarNumerosDeGrupo(){
          for [i = 0..49] {
               nro_grupo[i] = AsignarNroGrupo();
               signal(nro_grupo_asignado[i]);
          }
     }

     Procedure corregirTareas() {
          int alumnos_finalizados_por_grupo[1..25] = ([1..25] 0);
          int puntajeActual = 25, nro_grupo_tarea_completada;
          for [i = 0..49] {
               if(cant_tareas_completadas == 0)
                    wait(tarea_completada);
               }
               cant_tareas_completadas --;


cant_tareas_completadas.dequeue(nro_grupo_tarea_completada);

alumnos_finalizados_por_grupo[nro_grupo_tarea_completada] ++;


if(alumnos_finalizados_por_grupo[nro_grupo_tarea_completada] == 2)
{
                    notas[nro_grupo_tarea_completada] =
puntajeActual;
```

```
                    puntajeActual --;

signal(nota_lista[nro_grupo_tarea_completada]);

signal(nota_lista[nro_grupo_tarea_completada]);
                }
           }
     }
}

Process Alumno [id = 0..49] {
     int nro_grupo, nota;
     Comision.hacerFila(id, nro_grupo);
     hacerTarea(nro_grupo);
     Comision.avisarFinTarea(nro_grupo, nota);
}

Process JTP {
     Comision.esperarATodos();
     Comision.asignarNumerosDeGrupo();
     Comision.corregirTareas();
}
```

7)

```
7) - Versión 1

Monitor Maraton {
     int cant_corredores_en_el_inicio = 0, cant_botellas = 20;
     bool libre = true;
     cond preparados_listos_ya, no_hay_botellas, maquinaLibre,
hay_botellas;

     Procedure inicio() {
          cant_corredores_en_el_inicio ++;
          if (cant_corredores_en_el_inicio == C) {
               signal_all(preparados_listos_ya);
          } else {
               wait(preparados_listos_ya);
          }
     }

     Procedure fin() {
          if(!cola_botellas.empty()) {
               cola.enqueue(0);
               wait(maquinaLibre);
          } else {
               cola.enqueue(0);
          }

          if(cant_botellas == 0) {
               signal(no_hay_botellas);
               wait(hay_botellas);
          }
          cola.dequeue();
```

```
            sacarBotella();
            cant_botellas --;

            if(!cola.empty()) {
                  signal(maquinaLibre);
            }
      }

      Procedure reponerBotellas() {
            wait(no_hay_botellas);
      }

      Procedure botellasRepuestas() {
            cant_botellas = 20;
            signal(hay_botellas);
      }
}

Process Corredor [id = 0..C-1] {
      Maraton.inicio();
      correr();
      Maraton.fin();
}

Process Repositor {
      while(true) {
            Maraton.reponerBotellas();
            recargarMaquina();
            Maraton.botellasRepuestas();
      }
```

```
}
```

7) Versión 2

```
7) - Versión 2

Monitor Maraton {
	int cant_corredores_en_el_inicio = 0;
	cond preparados_listos_ya;

	Procedure inicio() {
		cant_corredores_en_el_inicio ++;
		if (cant_corredores_en_el_inicio == C) {
			signal_all(preparados_listos_ya);
		} else {
			wait(preparados_listos_ya);
		}
	}
}

Monitor AccesoAMaquina {
	bool libre = true;
	int cant_esperando = 0;
	cond maquina_libre;

	Procedure solicitar_acceso() {
		if (!libre) {
			cant_esperando ++;
			wait(maquina_libre);
		} else {
			libre = false;
```

```
            }
      }

      Procedure liberar() {
            if(cant_esperando > 0) {
                  signal(maquina_libre);
            } else {
                  libre = true;
            }
      }
}

Monitor Maquina {
      int cant_botellas = 20;
      bool hay_corredor = false;
      cond hay_que_reponer, botellas_repuestas;

      Procedure tomar_botella() {
            if (cant_botellas == 0) {
                  hay_corredor = true;
                  signal(hay_que_reponer);
                  wait(botellas_repuestas);
            }
            agarrar_botella();
            cant_botellas --;
      }

      Procedure reponer_botellas() {
            if(!hay_corredor) {
                  wait(hay_que_reponer);
            }
```

```
            hay_corredor = false;
            cant_botellas = 20;
            signal(botellas_repuestas);
      }
}


Process Corredor [id = 0, C-1] {
      Maraton.inicio();
      correr();
      AccesoAMaquina.solicitar_acceso();
      Maquina.tomar_botella();
      AccesoAMaquina.liberar();
      beber_agua();
}

Process Repositor {
      while (true) {
            Maquina.reponer_botellas();
      }
}
```

8)

```
8)

Monitor Equipo [id = 0..3] {
      int cant_llegaron = 0, cancha_a_usar;
      cond llegaron_todos;

      Procedure llegar(cancha: out int) {
            cant_llegaron ++;
            if(cant_llegaron == 5) {
                  Admin.obtener_cancha(cancha_a_usar);
                  signalall(llegaron_todos);
            } else {
                  wait(llegaron_todos);
            }
            cancha = cancha_a_usar;
      }
}

Monitor Cancha [id = 0..1] {
      int cant_llegaron = 0;
      cond comienzo_partido, fin_partido;

      Procedure llegar() {
            cant_llegaron ++;
            if(cant_llegaron == 10) {
                  signal(comienzo_partido);
            }
            wait(fin_partido);
      }
```

```
      Procedure irse() {
            se_va();
      }

      Procedure iniciar_tiempo() {
            if(cant_llegaron < 10) {
                  wait(comienzo_partido);
            }
      }

      Procedure finalizar_tiempo() {
            signalall(fin_partido);
      }
}

Monitor Admin {
      int cant_equipos = 0;
      Procedure obtener_cancha(cancha: out int) {
            cancha = cant_equipos div 2;
            cant_equipos ++;
      }
}

Procedure Jugador [id = 0..19] {
      int cancha;
      Equipo[DarEquipo()].llegar(cancha);
      Cancha[cancha].llegar();
}

Procedure Reloj [id = 0..1] {
      cancha[id].iniciar_tiempo();
```

```
      delay(50min);
      cancha[id].finalizar_tiempo();
}
```

9)

```
9)

Monitor Aula {
      int cant_alumnos_llegaron = 0, nota_alumno[45];
      Cola alumnos_que_entregaron;
      cond llegaron_todos, examen_recibido, nota_enviada,
examen_entregado;

      Procedure llegar(){
            cant_alumnos_llegaron ++;
            if(cant_alumnos_llegaron == 45) {
                  signal(llegaron_todos);
            }
            wait(examen_recibido);
      }

      Procedure entregarExamen(examen: in Examen, id: in int, nota:
out int) {
            alumnos_que_entregaron.enqueue(id, examen);
            signal(examen_entregado);
            wait(nota_enviada);
            nota = nota_alumno[id];
      }
```

```
      Procedure repartirExamenes() {
            if(cant_alumnos_llegaron < 45) {
                  wait(llegaron_todos);
            }
            for [_ = 0..44] {
                  signal(examen_recibido);
            }
      }

      Procedure recibirExamen(id: in int, examen: out Examen) {
            if(alumnos_que_entregaron.empty()) {
                  wait(examen_entregado);
            }
            alumnos_que_entregaron.dequeue(id, examen);
      }

      Procedure enviarNota(id: in int, nota: in int) {
            nota_alumno[id] = nota;
            signal(nota_enviada);
      }
}

Process Alumno [id = 0..44] {
      int nota;
      Examen examen:

      Aula.llegar();
      hacerExamen(examen);
      Aula.entregarExamen(examen, id, nota);
}
```

```
Process Preceptor {
      Aula.repartirExamenes();
}

Process Profesora {
      int nota;
      Examen examen;

      for [_ = 0..44] {
            Aula.recibirExamen(id, examen);
            corregirExamen(examen, nota);
            Aula.enviarNota(id, nota);
      }
}
```

10)

```
10)

Monitor Juego {
      int cant_esperando = 0;
      bool libre = true, hay_que_desinfectar = true;
      cond juego_libre, desinfectar, desinfeccion_finalizada;

      Procedure solicitarJuego() {
            if (!libre) {
                  cant_esperando ++;
                  wait(juego_libre);
                  cant_esperando --;
            }
```

```
            hay_que_desinfectar = true;
            signal(desinfectar);
            libre = false;
            wait(desinfeccion_finalizada);
      }

      Procedure liberarJuego() {
            if (cant_esperando > 0) {
                  signal(juego_libre);
            } else {
                  libre = true;
            }
      }

      Procedure hayQueDesinfectar() {
            if(!hay_que_desinfectar) {
                  wait(desinfectar);
            }
            hay_que_desinfectar = false;
      }

      Procedure disponibleParaUsar() {
            signal(desinfeccion_finalizada);
      }
}

Process Persona [id = 0..N-1] {
      Juego.solicitarJuego();
      Usar_juego();
      Juego.liberarJuego();
}
```

```
Process Empleado {
      for [_ = 0..N-1] {
            Juego.hayQueDesinfectar();
            Desinfectar_Juego();
            Juego.disponibleParaUsar();
      }
}
```