Programación concurrente

Práctica 4 - Pasaje de mensajes

Pasaje de mensajes asincrónico

1)

a)

```
1) a)

chan pedidos(Pedido, int), resultado[N](Resultado);

Process Cliente [id = 0..N-1] {
      Pedido pedido = getPedido();
      Resultado res;
      send pedidos(pedido, id);
      receive resultado[id](res);
}

Process Empleado {
      Pedido pedido;
      int id_cliente;
      Resultado res;
      while (true) {
            receive pedidos(pedido, id_cliente);
            resolver_pedido(pedido, res);
            send resultado[id_cliente](res);
      }
}
```

b)

```
1) b)

...
```

```
Process Empleado [0..1] {
...
```

c)

```
1) c)

chan pedidos(Pedido, int), resultado[N](Resultado),
solicitudes(int), respuesta[2](Pedido, int);

Process Cliente [id = 0..N-1] {
      Pedido pedido = getPedido();
      Resultado res;
      send pedidos(pedido, id);
      receive resultado[id](res);
}

Process Coordinador {
      Pedido pedido;
      int id_empleado, id_cliente;
      while(true) {
            receive solicitudes(id_empleado);
            if(empty(pedidos)) {
                  id_cliente = -1;
            } else {
                  receive pedidos(pedido, id_cliente);
            }
            send respuesta[id_empleado](pedido, id_cliente);
      }
}

Process Empleado [id = 0..1] {
```

```
      Pedido pedido;
      int id_cliente;
      Resultado res;
      while (true) {
            send solicitudes(id);
            receive respuesta[id](pedido, id_cliente);
            if(id_cliente == -1) {
                  delay(15min);
            } else {
                  resolver_pedido(pedido, res);
                  send resultado[id_cliente](res);
            }
      }
}
```

2)

```
2)

chan obtener_caja(int), caja_mas_vacia[P](int), atencion[5](int,
Pedido), comprobante[P](Comprobante), caja_libre(int),
mensaje_enviado();

Process Cliente [id = 0..P] {
      int caja;
      Pedido pedido = getPedido();
      Comprobante comprobante;

      send obtener_caja(id);
      send mensaje_enviado();
```

```
      receive caja_mas_vacia[id](caja);

      send atencion[caja](id, pedido);
      receive comprobante[id](comprobante);

      send caja_libre(caja);
      send mensaje_enviado();
}

Process Coordinador {
      int id_cliente, cant_caja[5] = ([5] 0), caja;
      while(true) {
            receive mensaje_enviado();
            if
            -     (!empty(obtener_caja)) -> {
                        receive obtener_caja(id_cliente);
                        get_menor(cant_caja, caja); # Devuelve en
caja el id de la caja mas vacia
                        cant_caja[caja] ++;
                        send caja_mas_vacia[id_cliente](caja);
                  }
            -     (!empty(caja_libre)) -> {
                        receive caja_libre(caja);
                        cant_caja[caja] --;
                  }
      }
}

Process Cajero [id = 0..4] {
      int id_cliente;
      Pedido pedido;
```

```
      Comprobante comprobante;

      while(true) {
            receive atencion[id](id_cliente, pedido);
            resolver_pedido(pedido, comprobante);
            send comprobante[id_cliente](comprobante);
      }
}
```

3)

```
3)

chan pedidos(Pedido, int), comidas[C](Comida),
pedidos_cocina(Pedido, int), solicitudes(int), respuesta[3]();

Procedure Cliente [id = 0..C-1] {
      Pedido pedido = getPedido();
      Comida comida;

      send pedidos(pedido, id);
      receive comida[id](comida);
      comer();
}

Procedure Coordinador {
      Pedido pedido;
      int id_empleado, id_cliente;

      while (true) {
```

```
            receive solicitudes(id_empleado);
            if(empty(pedidos)) {
                  pedido = null;
                  id_cliente = -1;
            } else {
                  receive pedidos(pedido, id_cliente);
            }
            send respuesta[id_empleado](pedido, id_cliente);
      }
}

Procedure Vendedor [id = 0..2] {
      Pedido pedido;
      int id_cliente;

      while(true) {
            send solicitudes(id);
            receive respuesta[id](pedido, id_cliente);
            if (id_cliente == -1) {
                  delay(rand(1, 3)*60);
            } else {
                  send pedidos_cocina(pedido, id_cliente);
            }
      }
}

Procedure Cocinero [id = 0..1] {
      Pedido pedido;
      int id_cliente;
      Comida comida;
```

```
      while(true) {
            receive pedidos_cocina(pedido, id_cliente);
            cocinar(pedido, comida);
            send comidas[id_cliente](comida);
      }
}
```

4)

a) y b)

```
4) a) y b)

chan solicitar_cabina(int), cabina_a_usar[N](int), pagar(int),
obtener_ticket[N](Ticket), mensaje_enviado();

Process Cliente [id = 0..N-1] {
      int cabina;
      Ticket ticket;

      send solicitar_cabina(id);
      send mensaje_enviado();
      receive cabina_a_usar[id](cabina);

      usar_cabina(cabina);

      send pagar(id, cabina);
      send mensaje_enviado();
      receive obtener_ticket[id_cliente](ticket);
}
```

```
Process Empleado {
      int id_cliente, cabina;
      Ticket ticket;
      bool cabinas_libres[10] = ([10] true)

      while(true) {
            receive mensaje_enviado();
            # algunoEsDistintoDeCero devuelve true si alguna
            # posicion es true
            if (empty(pagar) && algunoEsTrue(cabinas_libres)) {
                  receive solicitar_cabina(id_cliente);
                  get_cabina_libre(cabinas_libres, cabina);
                  cabinas_libres[cabina] = false;
                  send cabina_a_usar(cabina);
            } else {
                  receive pagar(id_cliente, cabina);
                  cabinas_libres[cabina] = true;
                  Cobrar(id_cliente, ticket);
                  send obtener_ticket[id_cliente](ticket);
            }
      }
}
```

5)

a)

```
5) a)

chan documentos(Documento);

Process Administrativo [id = 0..N-1] {
      Documento documento;
      bool tiene_documento_para_imprimir;

      while(true) {
            # tiene_documento_para_imprimir se pone en true cada
tanto
            if(tiene_documento_para_imprimir) {
                  documento = getDocumento();
                  send documentos(documento);
            }
      }
}

Process Impresora [id = 0..2] {
      Documento documento;

      while(true) {
            receive documentos(documento);
            Imprimir(documento);
      }
}
```

b)

```
5) b)

chan documentos(Documento), documentos_prioritarios(Documento),
imprimir(Documento), mensaje_enviado();

Process Administrativo [id = 0..N-1] {
      Documento documento;
      bool tiene_documento_para_imprimir;

      while(true) {
            # tiene_documento_para_imprimir se pone en true cada
tanto
            if(tiene_documento_para_imprimir) {
                  documento = getDocumento();
                  send documentos(documento);
                  send mensaje_enviado();
            }
      }
}

Process Director {
      Documento documento;
      bool tiene_documento_para_imprimir;

      while(true) {
            # tiene_documento_para_imprimir se pone en true cada
tanto
            if(tiene_documento_para_imprimir) {
                  documento = getDocumento();
                  send documentos_prioritarios(documento);
                  send mensaje_enviado();
```

```
            }
      }
}

Process Coordinador {
      Documento documento;

      while(true) {
            receive mensaje_enviado();

            if (empty(documentos_prioritarios)) {
                  receive documentos(documento);
                  send imprimir(documento);
            } else {
                  receive documentos_prioritarios(documento);
                  send imprimir(documento);
            }
      }
}

Process Impresora [id = 0..2] {
      Documento documento;

      while(true) {
            receive imprimir(documento);
            Imprimir(documento);
      }
}
```

c)

```
5) c)

chan documentos(Documento), imprimir(Documento);

Process Administrativo [id = 0..N-1] {
      Documento documento;
      bool tiene_documento_para_imprimir;
      int documentos_restantes = 10

      while(documentos_restantes != 0) {
            # tiene_documento_para_imprimir se pone en true cada
tanto
            if(tiene_documento_para_imprimir) {
                  documentos_restantes --;
                  documento = getDocumento();
                  send documentos(documento);
            }
      }
}

Process Coordinador {
      Documento documento;
      int documentos_recibidos = 0;

      for [_ = 0..(10*N)-1] {
            receive documentos(documento);
            send imprimir(documento);
      }

      for [_ = 0..2] {
            send imprimir(null);
```

```
      }
}

Process Impresora [id = 0..2] {
      Documento documento;
      bool termino = false;

      while(!termino) {
            receive imprimir(documento);
            if documento != null {
                  Imprimir(documento);
            } else {
                  termino = true;
            }
      }
}
```

d)

```
5) d)

chan documentos(Documento), imprimir(Documento),
mensaje_enviado();

Process Administrativo [id = 0..N-1] {
      Documento documento;
      bool tiene_documento_para_imprimir;
      int documentos_restantes = 10

      while(documentos_restantes != 0) {
            # tiene_documento_para_imprimir se pone en true cada
tanto
```

```
            if(tiene_documento_para_imprimir) {
                  documentos_restantes --;
                  documento = getDocumento();
                  send documentos(documento);
                  send mensaje_enviado();
            }
      }
}

Process Director {
      Documento documento;
      bool tiene_documento_para_imprimir;
      int documentos_restantes = 10

      while(documentos_restantes != 0) {
            # tiene_documento_para_imprimir se pone en true cada
tanto
            if(tiene_documento_para_imprimir) {
                  documentos_restantes --;
                  documento = getDocumento();
                  send documentos_prioritarios(documento);
                  send mensaje_enviado();
            }
      }
}

Process Coordinador {
      Documento documento;
      int documentos_recibidos = 0;

      for [_ = 0..(10*(N+1))-1] {
```

```
            receive mensaje_enviado();

            if (empty(documentos_prioritarios)) {
                  receive documentos(documento);
                  send imprimir(documento);
            } else {
                  receive documentos_prioritarios(documento);
                  send imprimir(documento);
            }
      }

      for [_ = 0..2] {
            send imprimir(null);
      }
}

Process Impresora [id = 0..2] {
      Documento documento;
      bool termino = false;

      while(!termino) {
            receive imprimir(documento);
            if documento != null {
                  Imprimir(documento);
            } else {
                  termino = true;
            }
      }
}
```

e) El inciso d) está resuelto sin busy waiting

Pasaje de mensajes sincrónico

1)

a) Procesos: Analizador y Robot
Recursos: direcciones de los sitios web posiblemente infectados.
Comunicaciones: Analizador -> Robot para avisar las direcciones

b)

```
1) b)

Process Robot [id = 0..R-1] {
      Text direccion;
      while(true) {
            busca_sitio_web_infectado(direccion);
            Analizador!web(direccion);
      }
}

Process Analizador {
      Text direccion;

      while(true) {
            Robot[*]?web(direccion);
            analizar(direccion);
      }
}
```

c)

```
1) c)

Process Robot [id = 0..R-1] {
      Text direccion;
      while(true) {
            busca_sitio_web_infectado(direccion);
            Admin!web(direccion);
      }
}

Process Admin {
      Cola<Text> direcciones;
      Text aux;

      do {
            -     Robot[*]?web(direccion) -> {
                        direcciones.push(direccion);
                  }
            -     !direcciones.empty(); Analizador?listo() -> {
                        direcciones.pop(aux);
                        Analizador!web(aux);
                  }
      }
}

Process Analizador {
      Text direccion;

      while(true) {
            Admin!listo()
            Admin?web(direccion);
```

```
            analizar(direccion);
      }
}
```

2)

```
2)

Process Empleado [id = 0..2] {
      if(id == 0) {
            ADN adn;

            while(true) {
                  preparar_muestra(adn);
                  Buffer!bufferear(adn);
            }
      } else if (id == 1) {
            ADN adn;
            Analisis set_de_analisis;
            Resultados resultado;

            while(true) {
                  Buffer!listo();
                  Buffer?muestra(adn);
                  armar_analisis(adn, set_de_analisis);
                  Empleado[2]!analisis(set_de_analisis);
                  Empleado[2]?res(resultado);
                  archivar(resultado);
            }
      } else {
```

```
            Analisis set_de_analisis;
            Resultados resultado;

            while(true) {
                  Empleado[1]?analisis(set_de_analisis);
                  realizar_analisis(set_de_analisis, resultado);
                  Empleado[1]!res(resultado);
            }
      }
}

Process Buffer {
      Cola<ADN> muestras;
      ADN adn;
      do
      -     Empleado[0]?bufferear(adn) -> {
                  muestras.push(adn);
            }
      -     !muestras.empty(); Empleado[1]?listo() -> {
                  Empleado[1]!muestra(muestras.pop());
            }
}
```

3)

a)

```
3) a)

Process Alumno [id = 0..N-1] {
      Examen examen = getExamen();
      int nota;

      resolver(examen);

      Admin!entregar(examen, id);
      Profesor?correccion(nota);
}

Process Admin {
      Cola<Examen> examenes;
      int id_alumno, cant_alumnos_entregaron;

      do {
            -     cant_alumnos_entregaron < N;
Alumno[*]?entregar(examen, id_alumno) -> {
                        examenes.push(examen, id_alumno);
                        cant_alumnos_entregaron ++;
                  }
            -     !examenes.empty(); Profesor?listo() -> {
                        examenes.pop(examen, id_alumno);
                        Profesor!corregir(examen, id_alumno);
                  }
      }
}

Process Profesor {
      Examen examen;
```

```
      int nota, id_alumno;

      for [_ = 0..N-1] {
            Admin!listo();
            Admin?corregir(examen, id_alumno);
            corregir(examen, nota);
            Alumno[id_alumno]!correccion(nota);
      }
}
```

b)

```
3) b)

Process Alumno [id = 0..N-1] {
      Examen examen = getExamen();
      int nota;

      resolver(examen);

      Admin!entregar(examen, id);
      Profesor[*]?correccion(nota);
}

Process Admin {
      Cola<Examen> examenes;
      int id_alumno, id_profesor, cant_alumnos_entregaron;

      do {
            -     cant_alumnos_entregaron < N;
Alumno[*]?entregar(examen, id_alumno) -> {
                        examenes.push(examen, id_alumno);
```

```
                    cant_alumnos_entregaron ++;
                }
          -     !examenes.empty(); Profesor[*]?listo(id_profesor)
-> {
                    examenes.pop(examen, id_alumno);
                    Profesor[id_profesor]!corregir(examen,
id_alumno);
                }
     }
     for [_ = 0..P-1] {
          Profesor[*]?listo(id_profesor);
          Profesor[id_profesor]!corregir(null, -1);
     }
}

Process Profesor [id = 0..P-1] {
     Examen examen;
     int nota, id_alumno;
     bool termino = false;

     while (!termino) {
          Admin!listo(id);
          Admin?corregir(examen, id_alumno);
          if (id_alumno == -1) {
               termino == true;
          } else {
               corregir(examen, nota);
               Alumno[id_alumno]!correccion(nota);
          }
     }
}
```

c)

```
3) c)

Process Alumno [id = 0..N-1] {
     Examen examen = getExamen();
     int nota;

     Admin!llegar_aula();
     Admin?empezar();

     resolver(examen);

     Admin!entregar(examen, id);
     Profesor[*]?correccion(nota);
}

Process Admin {
     Cola<Examen> examenes;
     int id_alumno, id_profesor, cant_alumnos_entregaron,
cant_alumnos_llegaron;

     do {
          -    cant_alumnos_llegaron < N; Alumno[*]?llegar_aula()
-> {
                    cant_alumnos_llegaron++;
               }
     }

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

```
	}

	do {
		-	cant_alumnos_entregaron < N;
Alumno[*]?entregar(examen, id_alumno) -> {
			examenes.push(examen, id_alumno);
			cant_alumnos_entregaron ++;
		}
		-	!examenes.empty(); Profesor[*]?listo(id_profesor)
-> {
			examenes.pop(examen, id_alumno);
			Profesor[id_profesor]!corregir(examen,
id_alumno);
		}
	}
	for [_ = 0..P-1] {
		Profesor[*]?listo(id_profesor);
		Profesor[id_profesor]!corregir(null, -1);
	}
}

Process Profesor [id = 0..P-1] {
	Examen examen;
	int nota, id_alumno;
	bool termino = false;

	while (!termino) {
		Admin!listo(id);
		Admin?corregir(examen, id_alumno);
		if (id_alumno == -1) {
			termino == true;
```

```
            } else {
                  corregir(examen, nota);
                  Alumno[id_alumno]!correccion(nota);
            }
      }
}
```

4)

a)

```
4) a)

Process Persona [id = 0..P-1] {
      Empleado!solicitar_acceso(id);
      jugar();
      Empleado!fin_juego();
}

Process Empleado {
      int id_persona;

      for [_ = 0..P-1] {
            Persona[*]?solicitar_acceso(id_persona);
            Persona[id_persona]?fin_juego();
      }
}
```

b)

```
4) b)

Process Persona [id = 0..P-1] {
     Admin!solicitar_acceso(id);
     Empleado?comenzar_juego();
     jugar();
     Empleado!fin_juego();
}

Process Admin {
     int id_persona;
     Cola<int> personas;

     do {
          -    Persona[*]?solicitar_acceso(id_persona) -> {
                    personas.push(id_persona);
               }
          -    !personas.empty(); Empleado?listo(); -> {
                    personas.pop(id_persona);
                    Empleado!solicitar_acceso(id_persona);
               }
     }
}

Process Empleado {
     int id_persona;

     for [_ = 0..P-1] {
          Admin!listo();
          Admin?solicitar_acceso(id_persona);
          Persona[id_persona]!comenzar_juego();
```

```
            Persona[id_persona]?fin_juego();
      }
}
```

5)

```
5)

Process Espectador [id = 0..E-1] {
      Admin!solicitar_acceso(id);
      Maquina?pasar();
      tomar_botella();
      Maquina!fin();
}

Process Admin {
      Cola<int> espectadores;

      do {
            -     Espectador[*]?solicitar_acceso(id_espectador) -> {
                        espectadores.push(id_espectador);
                  }
            -     !espectadores.empty(); Maquina?listo() -> {
                        Maquina!proximo(espectadores.pop());
                  }
      }
}

Process Maquina {
      int id_espectador;
```

```
      for [_ = 0..E-1] {
            Admin!listo();
            Admin?proximo(id_espectador);
            Espectador[id_espectador]!pasar();
            Espectador[id_espectador]?fin();
      }
}
```