Programación concurrente

Práctica 2 - Variables compartidas

1)

a) Se necesita un proceso persona para cada persona, el recurso de acceso exclusivo va a ser el detector de metales, el cual se va a acceder con exclusión mutua regulada mediante un semáforo

b)

```
1) b)

sem mutex = 1;

Process Persona [i=1..N] {
      P(mutex);
      pasar_por_detector();
      V(mutex);
}
```

c)

```
1) c)

sem mutex = 3;

Process Persona [i=1..N] {
      P(mutex);
      pasar_por_detector();
      V(mutex);
}
```

d)

```
1) d)

sem mutex = 3;

Process Persona [id=1..N] {
      int cantidad_pasos = rand();
      for (i = 1..cantidad_pasos){
            P(mutex);
            pasar_por_detector();
            V(mutex);
      }
}
```

2)

a)

```
2) a)

sem mutex = 1;
int ultimo_fallo = 0;
Fallo historial[N];

Process Proceso [id = 0..3] {
      P(mutex);
      while (historial.length() > 0){
            Fallo fallo = historial[ultimo_fallo];
            ultimo_fallo++;
            V(mutex);
            if(fallo.id() == 3){
                  print(fallo.toString());
```

```
            }
            P(mutex);
      }
      V(mutex);
}
```

b)

```
2) b)

sem mutex = 1, mutex2 = 1;
int ultimo_fallo = 0;
Fallo historial[N];
int cantidad_fallos[4] = ([4] 0)

Process Proceso [id = 0..3] {
      P(mutex);
      while (historial.length() > 0){
            Fallo fallo = historial[ultimo_fallo];
            ultimo_fallo++;
            V(mutex);
            P(mutex2);
            cantidad_fallos[fallo.id()]++;
            V(mutex2);
            P(mutex);
      }
      V(mutex);
}
```

c)

```
2) c)

sem mutex = 1;
int ultimo_fallo = 0;
Fallo historial[N];
int cantidad_fallos[4] = ([4] 0)

Process Proceso [id = 0..3] {
      P(mutex);
      while (historial.length() > 0){
            Fallo fallo = historial[ultimo_fallo];
            ultimo_fallo++;
            V(mutex);
            if(fallo.id() == id){
                  cantidad_fallos[fallo.id()]++;
            }
            P(mutex);
      }
      V(mutex);
}
```

3)

```
3)

sem mutex = 1, libre = 5;
Recurso cola[5];

Process Proceso [id = 1..P] {
      while(true){
```

```
            P(libre);
            P(mutex);
            Recurso recurso = cola.dequeue();
            V(mutex);

            usar_recurso(recurso);

            P(mutex);
            cola.enqueue(recurso);
            V(mutex);
            V(libre);
        }
}
```

4)

La solución presentada no es la más adecuada porque permite que al haber muchos procesos de un tipo de prioridad, estos hagan P(total) pero, una vez superado el límite para dicho tipo de proceso (4 o 5), sigan haciendo P(total) y hagan P(alta o baja), quedándose esperando en ese semáforo, pero ya habiendo hecho P(total), por lo que el otro tipo de prioridad podría no poder entrar al ser total = 0.

Una solución simple sería intercambiar la toma de los semáforos, haciendo P(alta); y luego P(total); y P(baja); y luego P(total); en el otro caso.

5)

```
5) a)

sem mutex = 1, no_vacio = 0, vacio = N;

Paquete contenedores[N];

Process Preparador {
      int libre = 0;
      while(true) {
            Paquete paquete = preparar_paquete();
            P(vacio);
            P(mutex);
            contenedores[libre] = paquete;
            V(mutex);
            V(no_vacio);
            libre = (libre + 1) mod N;
      }
}

Process Entregador {
      int ocupado = 0;
      while(true) {
            P(no_vacio);
            P(mutex);
            Paquete paquete = contenedores[ocupado];
            V(mutex);
            V(vacio);
            ocupado = (ocupado + 1) mod N;
            realizar_entrega(paquete);
```

```
      }
}
```

```
5) b)

sem mutex = 1, no_vacio = 0, vacio = N;

int libre = 0;

Paquete contenedores[N];

Process Preparador [id = 1..P] {
      while(true) {
            Paquete paquete = preparar_paquete();
            P(vacio);
            P(mutex);
            contenedores[libre] = paquete;
            libre = (libre + 1) mod N;
            V(mutex);
            V(no_vacio);
      }
}

Process Entregador {
      int ocupado = 0;
      while(true) {
            P(no_vacio);
            P(mutex);
            Paquete paquete = contenedores[ocupado];
            V(mutex);
            V(vacio);
```

```
            ocupado = (ocupado + 1) mod N;
            realizar_entrega(paquete);
      }
}
```

```
5) c)

sem mutex = 1, no_vacio = 0, vacio = N;

int ocupado = 0;

Paquete contenedores[N];

Process Preparador {
      int libre = 0;
      while(true) {
            Paquete paquete = preparar_paquete();
            P(vacio);
            P(mutex);
            contenedores[libre] = paquete;
            V(mutex);
            V(no_vacio);
            libre = (libre + 1) mod N;
      }
}

Process Entregador [id = 1..E] {
      while(true) {
            P(no_vacio);
            P(mutex);
            Paquete paquete = contenedores[ocupado];
```

```
            ocupado = (ocupado + 1) mod N;
            V(mutex);
            V(vacio);
            realizar_entrega(paquete);
      }
}
```

```
5) d)

sem mutex = 1, no_vacio = 0, vacio = N;

int ocupado = 0;
int libre = 0;

Paquete contenedores[N];

Process Preparador [id = 1..P] {
      while(true) {
            Paquete paquete = preparar_paquete();
            P(vacio);
            P(mutex);
            contenedores[libre] = paquete;
            libre = (libre + 1) mod N;
            V(mutex);
            V(no_vacio);
      }
}

Process Entregador [id = 1..E] {
      while(true) {
            P(no_vacio);
```

```
            P(mutex);
            Paquete paquete = contenedores[ocupado];
            ocupado = (ocupado + 1) mod N;
            V(mutex);
            V(vacio);
            realizar_entrega(paquete);
      }
}
```

6)

```
6) a)

sem impresora = 1;

Process Persona [id = 1..P] {
      Documento documento = get_documento(id);
      P(impresora);
      Imprimir(documento);
      V(impresora);
}
```

```
6) b)

sem mutex = 1, turno_de[P] = ([P] 0);

int cola[P] = [];
bool libre = true;

Process Persona [id = 1..P] {
      Documento documento = get_documento(id);
      P(mutex);
      if(libre){
            libre = false;
      } else {
            cola.enqueue(id);
            V(mutex);
            P(turno_de[id]);
      }

      Imprimir(documento);

      P(mutex);
      if(!cola.empty()){
            int prox_id = cola.dequeue();
            V(turno_de[prox_id]);
      } else {
            libre = true;
      }
      V(mutex);
}
```

```
6) c)

sem turno_de[P] = ([0] 1, [1..P-1] 0);

Process Persona [id = 0..P-1] {
      Documento documento = get_documento(id);

      P(turno_de[id]);
      Imprimir(documento);
      V(turno_de[(id + 1) mod P]);
}
```

```
6) d)

sem mutex = 1, turno_de[P] = ([P] 0), avisa_coordinador = 0,
termino_uso = 0;

int cola[P] = [];

Process Persona [id = 1..P] {
      Documento documento = get_documento(id);

      P(mutex)
      cola.enqueue(id);
      V(mutex)

      V(avisa_coordinador);
      P(turno_de[id]);
```

```
      Imprimir(documento);

      V(termino_uso);
}

Process Coordinador {
      while (true){
            P(avisa_coordinador);

            P(mutex);
            V(turno_de[cola.dequeue()]);
            V(mutex);

            P(termino_uso);
      }
}
```

```
6) e)

sem mutex = 1, mutex_impresora_libre = 1, turno_de[P] = ([P] 0),
avisa_coordinador = 0, impresoras = 5;

int cola[P] = [];
int impresora_libre[5] = ([5] true)
int impresora_a_usar[P] = ([P] -1);

Process Persona [id = 1..P] {
      Documento documento = get_documento(id);
```

```
      P(mutex)
      cola.enqueue(id);
      V(mutex);

      V(avisa_coordinador);
      P(turno_de[id]);

      Imprimir(documento, impresora_a_usar[id]);

      P(mutex_impresora_libre);
      impresora_libre[nro_impresora] = true;
      V(mutex_impresora_libre);

      V(impresoras);
}

Process Coordinador {
      int id, imp;
      while (true){
            P(avisa_coordinador);
            P(impresoras);
            for [i = 1..5] {
                  P(mutex_impresora_libre);
                  if(impresora_libre[i]){
                        imp = i;
                  }
                  V(mutex_impresora_libre);
            }
            impresora_libre[imp] = false;
```

```
            P(mutex);
            id = cola.dequeue();
            impresora_a_usar[id] = imp;
            V(turno_de[id]);
            V(mutex);
      }
}
```

7)

```
7)

sem mutex = 1, mutex_cola_fin = 1, eligieron_tarea = 0, comenzar =
0, grupo[10] = ([10] 0), avisar_profesor = 0;

int cant_eligieron_tarea = 0;
int cola_fin[50];
int puntajes[10];

Process Alumno [id = 1..50] {
      int tarea = elegir();

      P(mutex);
      cant_eligieron_tarea ++;
      if(cant_eligieron_tarea == 50){
            V(eligieron_tarea);
      }
      V(mutex);
```

```
      P(comenzar);

      realizar_tarea();

      P(mutex_cola_fin);
      cola_fin.enqueue(tarea);
      V(mutex_cola_fin);

      V(avisar_profesor);

      P(grupo[tarea]);

      ver_puntaje(puntajes[tarea]);
}

Process Profesor {
      int tarea;
      int cantidad_por_tarea[10] = ([10] 0);
      int orden_fin = 0;

      P(eligieron_tarea);
      for (i = 1..50){
            V(comenzar);
      }

      for (i = 1..50){
            P(avisar_profesor);

            P(mutex_cola_fin);
            tarea = cola_fin.dequeue();
            V(mutex_cola_fin);
```

```
            cantidad_por_tarea[tarea] ++;
            if(cantidad_por_tarea[tarea] == 5){
                  orden_fin ++;
                  puntajes[tarea] = orden_fin;
                  for (_ = 1..5) {
                        V(grupo[tarea]);
                  }
            }
      }
}
```

8)

```
8) a) y b)

sem mutex = 1, comenzar = 0;

int cant_empleados = 0;
int cant_piezas_dia = T;
int piezas_hechas = 0;
int cant_piezas_empleado[E] = ([E] 0)

Process Empleado [id = 1..E]{
      P(mutex);
      cant_empleados++;
      if(cant_empleados == E){
            V(comenzar);
      }
      V(mutex);
```

```
      P(comenzar);
      V(comenzar);

      P(mutex);
      while (piezas_hechas < cant_piezas_dia) {
            piezas_hechas ++;
            V(mutex);
            cant_piezas_empleado[id] ++;
            hacer_pieza();
            P(mutex);
      }
      V(mutex);
}
```

9)

```
9)

sem mutex_marco = 1, mutex_vidrio = 1, marcos_lleno = 0,
marcos_vacio = 30, vidrios_lleno = 0, vidrios_vacio = 50;

Marco deposito_marcos[30];
marco_libre = 0;
marco_ocupado = 0

Vidrio deposito_vidrios[50];
vidrio_ocupado = 0;

Process Carpintero [id = 0..3]{
```

```
     while (true){
          Marco marco = hacer_marco();

          P(marcos_vacio);
          P(mutex_marco);
          deposito_marcos[marco_libre] = marco;
          marco_libre = (marco_libre + 1) mod 30;
          V(mutex_marco);
          V(marcos_lleno);
     }
}

Process Vidriero {
     vidrio_libre = 0;
     while (true){
          Vidrio vidrio = hacer_vidrio();

          P(vidrios_vacio);
          P(mutex_vidrio);
          deposito_vidrios[vidrio_libre] = vidrio;
          vidrio_libre = (vidrio_libre + 1) mod 50;
          V(mutex_vidrio);
          V(vidrios_lleno);
     }
}

Process Armador [id = 0..2]{
     while (true){
          P(marcos_lleno);
          P(mutex_marco);
          Marco marco = deposito_marcos[marco_ocupado];
```

```
            marco_ocupado = (marco_ocupado + 1) mod 30;
            V(mutex_marco);
            V(marcos_vacio);

            P(vidrios_lleno);
            P(mutex_vidrio);
            Vidrio vidrio = deposito_vidrios[vidrio_ocupado];
            vidrio_ocupado = (vidrio_ocupado + 1) mod 50;
            V(mutex_vidrio);
            V(vidrios_vacio);

            armar_ventana(marco, vidrio);
        }
}
```

10)

a)

Versión 1 - Tiene un problema: el proceso coordinador, en el caso en que haya espacios totales libres, pero no haya más espacios para trigo/maíz, se queda en P(trigo/maiz) hasta que se libere un lugar de trigo/maiz, pudiendo haber camiones del otro tipo que deberían poder acceder, pero como el coordinador está esperando en el semáforo no podrán acceder.

```
10) a) - Versión 1

sem mutex = 1, turno_trigo = ([T] 0), turno_maiz = ([M] 0),
avisar_coordinador = 0, total = 7, trigo = 5, maiz = 5;

(string, int) cola[M + T];

Process CamionTrigo [id = 1..T]{
```

```
      P(mutex);
      cola.enqueue(("trigo", id))
      V(mutex);

      V(avisar_coordinador);
      P(turno_trigo[id]);
      descargar();
      V(total);
      V(trigo);
}

Process CamionMaiz [id = 1..M]{
      P(mutex);
      cola.enqueue(("maiz", id))
      V(mutex);

      V(avisar_coordinador);
      P(turno_maiz[id]);
      descargar();
      V(total);
      V(maiz);
}

Process Coordinador {
      while (true){
            P(avisar_coordinador);
            P(mutex);
            tipo, id = cola.dequeue();
            V(mutex);

            P(total);
```

```
            if(tipo == "trigo"){
                  P(trigo);
                  V(turno_trigo[id])
            }
            else if(tipo == "maiz"){
                  P(maiz);
                  V(turno_maiz[id])
            }
      }
}
```

Versión 2 - Tiene un problema: se podría producir busy waiting. En el caso en el que todos los espacios de un tipo de camión estén ocupados, pero siga habiendo espacios totales libres, y todos los camiones de la cola sean del tipo que está ocupado, el proceso coordinador va quedarse haciendo busy waiting, ya que va a desencolar el id que está en la punta de la cola 😏, y como hay espacios totales, pero no hay espacios del tipo solicitado, va a volver a encolar al final de la cola el id que previamente había desencolado, y hará eso una y otra vez hasta que haya una solicitud del tipo que tiene espacios libres, o hasta que se libere algún lugar del tipo que está completamente ocupado.

```
10) a) - Versión 2

sem mutex = 1, mutex_contador_trigo = 1, mutex_contador_maiz = 1,
turno_trigo = ([T] 0), turno_maiz = ([M] 0), avisar_coordinador =
0, total = 7;

(string, int) cola[M + T];
int cant_trigo_libre = 5, cant_maiz_libre = 5;

Process CamionTrigo [id = 1..T]{
      P(mutex);
      cola.enqueue(("trigo", id))
      V(mutex);
```

```
      V(avisar_coordinador);
      P(turno_trigo[id]);

      descargar();

      P(mutex_contador_trigo);
      cant_trigo_libre --;
      V(mutex_contador_trigo);
      V(total);
}

Process CamionMaiz [id = 1..M]{
      P(mutex);
      cola.enqueue(("maiz", id))
      V(mutex);

      V(avisar_coordinador);
      P(turno_maiz[id]);

      descargar();

      P(mutex_contador_maiz);
      cant_maiz_libre --;
      V(mutex_contador_maiz);
      V(total);
}

Process Coordinador {
      while (true){
            P(avisar_coordinador);
```

```
            P(mutex);
            tipo, id = cola.dequeue();
            V(mutex);

            P(total);

            if(tipo == "trigo" && cant_trigo_libre > 0){
                  P(mutex_contador_trigo);
                  cant_trigo_libre --;
                  V(mutex_contador_trigo);
                  V(turno_trigo[id]);
            }
            else if(tipo == "maiz" && cant_maiz_libre > 0){
                  P(mutex_contador_maiz);
                  cant_maiz_libre --;
                  V(mutex_contador_maiz);
                  V(turno_maiz[id]);
            } else {
                  cola.enqueue(tipo, id);
                  V(avisar_coordinador);
                  V(total);
            }
      }
}
```

b)

```
10) b)

sem e = 1, trigo = 5, maiz = 5, total = 7;

Process CamionTrigo [id = 1..T]{
     P(trigo);
     P(total);

     descargar();

     V(total);
     V(trigo);
}

Process CamionMaiz [id = 1..M]{
     P(maiz);
     P(total);

     descargar();

     V(total);
     V(maiz);
}
```

11)

```
11)

sem mutex = 1, avisar_empleado = 0, fue_vacunado[50]  = ([50] 0);

int cola[50];

Process Persona [id = 1..50] {
      P(mutex);
      cola.enqueue(id);
      V(mutex);

      V(avisar_empleado);
      P(fue_vacunado[id]);
}

Process Empleado {
      int grupo[5] = [];
      for(_ = 1..10){
            for (__ = 1..5){
                  P(avisar_empleado);
                  P(mutex);
                  grupo.push(cola.dequeue());
                  V(mutex);
            }

            for (persona in grupo){
                  vacunarPersona(persona);
            }
```

```
            for (persona in grupo){
                  V(fue_vacunado[persona]);
            }
            grupo.clean();
      }
}
```

12)

a)

```
sem mutex = 1, mutex_hispados = 1, mutex_cola_puesto[3] = ([3] 1),
avisar_recepcionista = 0, avisar_enfermera[3] = ([3] 0),
hisopar[150] = ([150] 0);

int cantidad_gente_puesto[3] = ([3] 0)
int cola_puesto[3][50];
int cola_inicial[150];
int puesto_a_usar;
int cantidad_pasajeros_hispados = 0;

Process Pasajero [id = 1..150] {
      P(mutex);
      cola_inicial.enqueue(id);
      V(mutex);

      V(avisar_recepcionista);
      # Hisopándose
      P(hisopar[id]);
      # Se retira
}
```

```
Process Recepcionista {
      int id;

      for (_ = 1..150){
            P(avisar_recepcionista);

            puesto_a_usar = min(cantidad_gente_puesto)
            cantidad_gente_puesto[puesto_a_usar] ++;

            P(mutex);
            id = cola_inicial.dequeue();
            V(mutex);

            P(mutex_cola_puesto[puesto]);
            cola_puesto[puesto].enqueue(id);
            V(mutex_cola_puesto[puesto]);
            V(avisar_enfermera[puesto]);
      }
}

Process Enfermera [id = 1..3] {
      int id;
      P(mutex_hispados);
      while (cantidad_pasajeros_hispados < 150) {
            cantidad_pasajeros_hispados ++;
            V(mutex_hispados)

            P(avisar_enfermera[id]);

            P(mutex_cola_puesto[id]);
```

```
            id = cola_puesto.dequeue();
            V(mutex_cola_puesto[id]);

            Hisopar(id);
            V(hisopar[id]);

            P(mutex_hispados);
      }
      V(mutex_hispados);
}
```

b)

```
sem mutex = 1, mutex_hispados = 1, mutex_cola_puesto[3] = ([3] 1),
avisar_enfermera[3] = ([3] 0), hisopar[150] = ([150] 0);

int cantidad_gente_puesto[3] = ([3] 0)
int cola_puesto[3][50];
int cantidad_pasajeros_hispados = 0;

Process Pasajero [id = 1..150] {
      int puesto;
      P(mutex);
            puesto = min(cantidad_gente_puesto)
            cantidad_gente_puesto[puesto] ++;
      V(mutex);

      P(mutex_cola_puesto[puesto]);
      cola_puesto[puesto].enqueue(id);
      V(mutex_cola_puesto[puesto]);

      V(avisar_enfermera[puesto]);
```

```
      # Hisopándose
      P(hisopar[id]);
      # Se retira
}

Process Enfermera [id = 1..3] {
      int id;
      P(mutex_hispados);
      while (cantidad_pasajeros_hispados < 150) {
            cantidad_pasajeros_hispados ++;
            V(mutex_hispados)

            P(avisar_enfermera[id]);

            P(mutex_cola_puesto[id]);
            id = cola_puesto.dequeue();
            V(mutex_cola_puesto[id]);

            Hisopar(id);
            V(hisopar[id]);

            P(mutex_hispados);
      }
      V(mutex_hispados);
}
```