Practica 3

2. Existen N procesos que deben leer información de una base de datos, la cual es administrada
   por un motor que admite una cantidad limitada de consultas simultáneas.
   a) Analice el problema y defina qué procesos, recursos y monitores/sincronizaciones
   serán necesarios/convenientes para resolverlo.
   b) Implemente el acceso a la base por parte de los procesos, sabiendo que el motor de
   base de datos puede atender a lo sumo 5 consultas de lectura simultáneas.

a) los procesos van a ser procesos, la bd es el recurso compartido, motor el monitor,

b)

```
process proceso [id: 0..N-1]{
  motor.acceder()
  motor.salir()
}

monitor Motor{

cola cond
limite = 5

procedure acceder(){

while (limite == 0){
    wait(cola)
```

```
}
limite--


}

procedure salir(){

signal(cola)
limite++

}
```

3. Existen N personas que deben fotocopiar un documento. La fotocopiadora sólo puede ser
usada por una persona a la vez. Analice el problema y defina qué procesos, recursos y
monitores serán necesarios/convenientes, además de las posibles sincronizaciones requeridas
para resolver el problema. Luego, resuelva considerando las siguientes situaciones:
a) Implemente una solución suponiendo no importa el orden de uso. Existe una función
Fotocopiar() que simula el uso de la fotocopiadora.
b) Modifique la solución de (a) para el caso en que se deba respetar el orden de llegada.
c) Modifique la solución de (b) para el caso en que se deba dar prioridad de acuerdo con la
edad de cada persona (cuando la fotocopiadora está libre la debe usar la persona de mayor
edad entre las que estén esperando para usarla).

d) Modifique la solución de (a) para el caso en que se deba respetar estrictamente el orden
dado por el identificador del proceso (la persona X no puede usar la fotocopiadora hasta
que no haya terminado de usarla la persona X-1).
e) Modifique la solución de (b) para el caso en que además haya un Empleado que le indica
a cada persona cuando debe usar la fotocopiadora.
f) Modificar la solución (e) para el caso en que sean 10 fotocopiadoras. El empleado le indica
a la persona cuál fotocopiadora usar y cuándo hacerlo.

a.

```
process Persona[id:0..N-1]{
    fotocopiadora.Fotocopiar()
}

Monitor fotocopiadora{
    procedure Fotocopiar()
}
```

b.

```
process Persona[id:0..N-1]{
    fotocopiadora.pedir()
    Fotocopiar()
    fotocopiadora.dejar()
}

Monitor fotocopiadora{
    cond cola
```

```
    esperando = 0
    bool libre = true

    procedure pedir(){

        if(!libre){
            esperando++
            wait(cola)
        }
        else
            libre = false
    }

    procedure dejar(){

        if (esperando > 0){
            esperando--
            signal(cola)
        }
        else
            libre = true
    }
```

c.

```
process Persona[id:0..N-1]{
    int edad, id
    fotocopiadora.pedir(edad,id)
    Fotocopiar()
    fotocopiadora.dejar()
}

Monitor fotocopiadora{
    bool libre = true
    cond cola[N]
    int idAux
```

```
    int esperando = 0
    colaOrdenada fila

    procedure pedir(idP,edad: in int){
        if (!libre){
            fila.insertarOrdenado(idAux, edad)
            esperando++
            wait(espera[idP])
        }
        else
            libre = false
    }

    procedure dejar(){
        if(esperando > 0){
            esperando--
            fila.sacar(idAux)
            signal(espera[idAux])
        {
        else
            libre = true
    }
End monitor}
```

d.

```
process Persona[id:0..N-1]{
    fotocopiadora.pedir(id)
    Fotocopiar()
    fotocopiadora.dejar()
}

Monitor fotocopiadora{

    int actual = 0
    cond cola[N]
```

```
    procedure pedir(id: in int){
        if (actual != id)
            wait(cola[id])
    }

    procedure dejar(){
        actual++
        signal(cola[actual])
    }
```

e.

```
process Persona[id:0..N-1]{
    fotocopiadora.pedir()
    Fotocopiar()
    fotocopiadora.dejar()
}

process Empleado{
    for i := 0 to N-1:
        fotocopiadora.atender()
}

Monitor fotocopiadora{
    cond esperarPersona, esperarFotocopiadora, esperarTurno
    int esperando = 0
    bool libre = true

    procedure atender(){
        if (esperando == 0)
            wait(esperarPersona)
        if (!libre)
```

```
            wait(esperarFotocopiadora)
        esperando--
        signal(esperarTurno)
    }

    procedure pedir(){
        esperando++
        signal(personas)
        wait(esperarTurno)
        libre=false
    }

    procedure dejar(){
        libre = true
        signal(esperarFotocopiadora)
    }
}
```

f.

```
Monitor fotocopiadora {
    cond esperarPersona, esperarFotocopiadora, esperarTurno
    cola personas[N] //cola en la que las personas encolan su id
    cola pedidoFotocopiadoras[10] //cola en la que se encolan lo
    int personaConFotocopiadora[N] //Vector indexado por id de p
    int esperando = 0

    for i := 0 to 9:
        pedidoFotocopiadoras.push(i)

    procedure pedir(id: IN int, fotocopiadoraUso: OUT int){
        esperando++
        personas.push(id)
        signal(esperarPersona)
```

```
        wait(esperarTurno)
        fotocopiadoraUso = personaConFotocopiadora[id]
    }

    procedure atender(){
        if (esperando == 0)
            wait(personas)
        if (pedidoFotocopiadoras.vacio())
            wait(esperarFotocopiadora)
        esperando--
        personaConFotocopiadora[personas.pop()] = pedidoFotocopi
        signal(esperarTurno)
    }

    procedure dejar(fotocopiadoraUso: IN int){
        pedidoFotocopiadoras.push(fotocopiadoraUso)
        signal(esperarFotocopiadora)
    }

END MONITOR}

process Persona[id:0..N-1]{
    int idFotocopiadora
    fotocopiadora.pedir(id,idFotocopiadora)
    Fotocopiar()
    fotocopiadora.dejar(idFotocopiadora)
}

process Empleado{
    for i := 0 to N-1:
        fotocopiadora.atender()
}
```

4. Existen N vehículos que deben pasar por un puente de acuerdo con el orden de llegada.
   Considere que el puente no soporta más de 50000kg y que cada vehículo cuenta con su propio
   peso (ningún vehículo supera el peso soportado por el puente).

```
Monitor puente{
    cond cola
    cola fila [N]
    pesoPuente = 0

    procedure entrar(miPeso : real in){
        if ((miPeso + pesoPuente > 50000) or (!fila.vacio()))
            fila.push(miPeso)
            wait(cola)
        else
            pesoPuente += miPeso
    }

    procedure salir(miPeso: real in){
        pesoPuente -= miPeso
        real pesoProx = fila.top()
        while (pesoProx + pesoPuente <= 50000) and (!fila.vacio
            pesoPuente += fila.pop()
            signal(cola)
            if (!fila.vacio())
                pesoProx = fila.top()
        }

END Monitor}

process Auto [id: 0..N-1]{
    puente.entrar(id.peso)
    puente.salir(id.peso)
```

```
}
```

5. En un corralón de materiales se deben atender a N clientes de acuerdo con el orden de llegada.
   Cuando un cliente es llamado para ser atendido, entrega una lista con los productos que
   comprará, y espera a que alguno de los empleados le entregue el comprobante de la compra
   realizada.
   a) Resuelva considerando que el corralón tiene un único empleado.
   b) Resuelva considerando que el corralón tiene E empleados (E > 1). Los empleados no
   deben terminar su ejecución.
   c) Modifique la solución (b) considerando que los empleados deben terminar su ejecución
   cuando se hayan atendido todos los clientes

a)

```
Monitor corralon{
    cond llamado, cliente, entregaComprobante, entregaLista
    bool empleadoListo = false
    int esperando= 0
    lista productos
    string comprobanteParaCliente

    procedure esperarLlamado(){
            if (!empleadoListo){
                esperando++
                wait(llamado)
            }
```

```
            signal(cliente)
            empleadoListo = false
    }

    procedure entregarLista(lista: in string){
        productos = lista
        signal(entregaLista)
    }

    procedure llamar(listaProd: out string){
        empleadoListo = true
        if (esperando > 0){
            esperando--
            signal(llamado)
        }
        else
            wait(cliente)
        wait(entregaLista)
        listaProd = productos
    }

    procedure darComprobante(comprobante: in string){
        comprobanteParaCliente = comprobante
        signal(entregaComprobante)
    }

    procedure recibirComprobante(comprobante: out string){
        wait(entreComprobante)
        comprobante = comprobanteParaCliente
    }




process Cliente[id: 0..N-1]{
```

```
    string lista
    string comprobante
    corralon.esperarLlamado()
    corralon.entregarLista(lista)
    corralon.recibirComprobante(comprobante)
}

process Empleado{
    lista string
    comprobante string
    for i := 0 to N-1
        corralon.llamar(lista)
        comprobante = generarComprobante(lista) //genera comprol
        corralon.darComprobante(comprobante)
}
```

b)

```
Monitor corralon{
    cola elibres
    cond esperaC
    int esperando = 0
    cantLibres = 0

    procedure Llegada(idE: out int){
        if (cantLibres == 0){
            esperando++
            wait(esperaC)
        }
        else
            cantLibres--
        pop(elibres)
    }
```

```
    procedure Proximo(idE: in int){
        push(elibres, idE)
        if (esperando > 0){
            esperando--
            signal(esperaC)
        }
        else
            cantLibres++
    }
END Monitor}


Monitor Escritorio[id:0..E-1]{

    cond vcCliente, vcEmpleado
    text lista,comprobanteParaCliente
    boolean listaCargada = false

    procedure atencion(listaProductos: in text, Comprobante: out
        lista = listaProductos
        listaCargada = true
        signal(vcEmpleado)
        wait(vcCliente)
        comprobante = comprobanteParaCliente
        signal(vcEmpleado)
    }

    procedure esperarLista(listaProductos: out text){
        if (!listaCargada)
            wait(vcEmpleado)
        listaProductos = lista
    }

    procedure darComprobante(comprobante : int text){
        comprobanteParaCliente=comprobante
        signal(vcCliente)
```

```
        wait(vcEmpleado)
        listaCargada = false
    }

    process Cliente[id:0..N-1]{
        int idE
        text lista, comprobante
        corralon.Llegada(idE)
        Escritorio[idE].atencion(lista,comprobante)
    }

    process Empleado[id:0..E-1]{
        text lista,comprobante
        while(true){
            corralon.proximo(id)
            Escritorio[id].esperarLista(lista)
            comprobante = generarComprobante(lista)
            Escritorio[id].darComprobante(comprobante)
        }
```

c)

```
Monitor corralon{
    cola elibres
    cond esperaC
    int esperando = 0
    cantLibres = 0

    procedure Llegada(idE: out int){
```

```
        if (cantLibres == 0){
            esperando++
            wait(esperaC)
        }
        else
            cantLibres--
        pop(elibres)
    }

    procedure Proximo(idE: in int){
        push(elibres, idE)
        if (esperando > 0){
            esperando--
            signal(esperaC)
        }
        else
            cantLibres++
    }
END Monitor}


Monitor Escritorio[id:0..E-1]{

    cond vcCliente, vcEmpleado
    text lista,comprobanteParaCliente
    boolean listaCargada = false

    procedure atencion(listaProductos: in text, Comprobante: ou
        lista = listaProductos
        listaCargada = true
        signal(vcEmpleado)
        wait(vcCliente)
        comprobante = comprobanteParaCliente
        signal(vcEmpleado)
    }
```

```
procedure esperarLista(listaProductos: out text){
    if (!listaCargada)
        wait(vcEmpleado)
    listaProductos = lista
}

procedure darComprobante(comprobante : int text){
    comprobanteParaCliente=comprobante
    signal(vcCliente)
    wait(vcEmpleado)
    listaCargada = false
}

process Cliente[id:0..N-1]{
    int idE
    text lista, comprobante
    corralon.Llegada(idE)
    Escritorio[idE].atencion(lista,comprobante)
}

process Empleado[id:0..E-1]{
    text lista,comprobante
    int atendidos = 0
    while(atendidos < N){
        corralon.proximo(id)
        Escritorio[id].esperarLista(lista)
        comprobante = generarComprobante(lista)
        Escritorio[id].darComprobante(comprobante)
    }
```

9)

En un examen de la secundaria hay un preceptor y una profesora que deben tomar un examen
escrito a 45 alumnos. El preceptor se encarga de darle el enunciado del examen a los alumnos

cundo los 45 han llegado (es el mismo enunciado para todos). La profesora se encarga de ir
corrigiendo los exámenes de acuerdo con el orden en que los alumnos van entregando. Cada
alumno al llegar espera a que le den el enunciado, resuelve el examen, y al terminar lo deja
para que la profesora lo corrija y le envíe la nota. Nota: maximizar la concurrencia; todos los
procesos deben terminar su ejecución; suponga que la profesora tiene una función
corregirExamen que recibe un examen y devuelve un entero con la nota.

```
Monitor Profe{


}

Monitor Prece{
    int llegados = 0
    cond esperaAlumnos
    cola alumnos
    text examenParaAlumno


    procedure llegue(id: int in){
        if (llegados < 45)
            llegados++
            cola.push(id)
            wait(esperaAlumnos)
        else
            signall(esperaAlumnos)
    }

    procedure recibirExamen(examen: text out){
        examenParaAlumno = examen
```

```
    procedure darExamen

process Alumno[id:0..44]{
    examen text
    Prece.llegue(id)
}

process Preceptor{
    for i:= 1 to 45:
        Prece.darExamen()
```

10

En un parque hay un juego para ser usada por N personas de a una a la vez y de acuerdo al
orden en que llegan para solicitar su uso. Además, hay un empleado encargado de desinfectar el
juego durante 10 minutos antes de que una persona lo use. Cada persona al llegar espera hasta
que el empleado le avisa que puede usar el juego, lo usa por un tiempo y luego lo devuelve.
Nota: suponga que la persona tiene una función Usar_juego que simula el uso del juego; y el
empleado una función Desinfectar_Juego que simula su trabajo. Todos los procesos deben
terminar su ejecución.

ASUMO QUE EL JUEGO VIENE LIMPIO

```
Monitor Juego {
    cond esperaEmpleado, esperaJugador;
    int esperando = 0;

    // Procedimiento que una persona llama para esperar a que el
    procedure esperarJugador() {
```

```
        if (esperando > 0) {
            esperando--; // Si hay personas esperando, una menos
        } else {
            wait(esperaJugador); // Esperar la señal del emplead
        }
    }

    // Procedimiento que el empleado llama para avisar que el ju
    procedure avisarEmpleado() {
        signal(esperaJugador); // Avisar a la persona que puede
    }

    // Procedimiento que una persona llama al llegar al juego
    procedure esperarJuego() {
        esperando++; // Aumentar el número de personas esperando
        wait(esperaEmpleado); // Esperar a que el empleado desin
    }

    // Procedimiento que el empleado llama para avisar que ha te
    procedure avisarJugador() {
        signal(esperaEmpleado); // Avisar a la siguiente persona
    }
}

Process personas [id: 0..N] {
    Juego.esperarJuego();   // Esperar a que el empleado desinfe
    Usar_juego();           // Simular el uso del juego
    Juego.avisarEmpleado(); // Avisar al empleado que ya termino
}

Process Empleado {
    for i := 1 to N do {
        Juego.esperarJugador();   // Esperar a que una persona l
        Desinfectar_Juego();      // Simular la desinfección del
        delay(10);                // Tiempo que tarda en desinfe
        Juego.avisarJugador();    // Avisar a la persona que ya
```

```
    }
}
```