Ejercicios de sincronización

1- Dados los siguientes procesos con variables compartidas, sincronizarlos para garantizar la mutua exclusión sobre ellas.

`variables_compartidas a = b = 1;`	
Proceso 0	**Proceso 1**

`variable_local d = 1;`
`While (TRUE){`
`  a = a + d;`
`  d = d * d;`
`  b = b – d;`
`}`
	`variable_local e = 2;`
	`While (TRUE){`
	`  b = b * e;`
	`  e = e ^ e;`
	`  a++;`
	`}`

2- Dado un sistema con N procesos del mismo programa, sincronice su código mediante semáforos para respetar los siguientes límites:

a. 3 recursos con M instancias
b. M recursos con 3 instancias

Programa

`while (TRUE){`
`  id_recurso = pedir_recurso();`
`  usar_recurso(id_recurso);`
`}`

3- Dado un sistema con los siguientes tipos de procesos, sincronice su código mediante semáforos sabiendo que hay tres impresoras, dos scanners y una variable compartida.

Proceso A (n instancias)	**Proceso B** (n instancias)	**Proceso C** (n instancias)

`While (TRUE){`
`  usar_impresora();`
`  variable_compartida++;`
`}`
	`While (TRUE){`
	`  variable_compartida++;`
	`  usar_scanner();`
	`}`
		`While (TRUE){`
		`  usar_scanner();`
		`  usar_impresora();`
		`}`

4- Sean dos procesos A y B, sincronizarlos para que ejecuten de manera alternada: A,B,A,B...

5- Sean los procesos A, B y C, sincronizarlos para que ejecuten de manera alternada: A,B,C,A,B,C...

6- Sean los procesos A, B y C, sincronizarlos para que ejecuten de la siguiente manera: B,A,C,A,B,A,C,A...

7- Suponga que un proceso tiene por tarea compilar un conjunto de programas y luego enviar el resultado de cada compilación por email al encargado de ese proyecto. Dicho proceso está organizado de la siguiente manera: N hilos de kernel compilan cada uno un programa distinto, y luego cada uno de ellos depositan en una lista (compartida para todo

el proceso) el resultado; por otro lado, un hilo de kernel retira los resultados de las compilaciones y manda un email por cada uno de ellos.

Estructura compartida: lista // Lista de resultados de compilaciones	
KLT compilador (N instancias)	KLT notificador (1 instancia)

While (TRUE){
id_programa = obtener_nuevo_programa();
r = compilar_programa(id_programa);
depositar_resultado(r, lista);
}
	While (TRUE){
	r2 = retirar_resultado(lista);
	enviar_email(r2);
	}

Asumiendo que la cantidad de programas a compilar es infinita, sincronice dicho código mediante el uso de semáforos para lograr un correcto uso de los recursos bajo los siguientes contextos:
a) Asumiendo que la lista no tiene límite de tamaño.
b) Asumiendo que la lista tiene un límite de M resultados como máximo.

8- Existe un aeropuerto que se utiliza como base de operaciones de una flota de aviones. Existen muchos aviones, diez pistas de aterrizaje / despegue y dos controladores aéreos. Cada vez que un avión desea despegar o aterrizar, debe utilizar una pista. Para ello, la misma es solicitada al controlador de entrada, y luego de ser utilizada se le notifica al controlador de salida para que vuelva a estar disponible.
Se pide que sincronice el siguiente pseudo-código respetando las reglas establecidas, sin que se produzca deadlock ni starvation (cuando el avión ya pidió pista). Para ello solamente debe utilizar semáforos, indicando el tipo de los mismos y sus valores iniciales.

pistasLibres = 10; // variable compartida		
AVIÓN	**CONTROLADOR ENTRADA**	**CONTROLADOR SALIDA**

while(TRUE){
mantenimiento();
despegar();
volar();
aterrizar();
}
	while(TRUE){
	otorgarUnaPista();
	pistasLibres--;
	log(pistasLibres);
	}
		while(TRUE){
		liberarUnaPista();
		pistasLibres++;
		log(pistasLibres);
		}

Nota: La función log() imprime por pantalla el valor actual de pistas libres.		

9- Se tiene un programa para simular la ejecución de penales de un partido de fútbol, el cual consta de tres procesos: árbitro, jugador y arquero. El pseudo-código es el siguiente:

Proceso ÁRBITRO	Proceso JUGADOR	Proceso ARQUERO

while(TRUE){
dar_orden();
validar_tiro();
}
	while(TRUE){
	posicionarse();
	patear();
	if (GOL==TRUE){
	festejar();
	else{
	lamentarse();
	}
		while(TRUE){
		posicionarse();
		atajar();
		if (GOL==FALSE){
		festejar();
		else{
		lamentarse();
		}

Las reglas que se deben cumplir son las siguientes:
Existen cinco procesos jugadores, un proceso árbitro y un proceso arquero.
Los jugadores no pueden patear si el árbitro no lo indicó.
El arquero no puede atajar si el jugador no pateó.
El árbitro sólo puede dar la orden cuando el jugador y el arquero están posicionados.
Existe una variable global GOL, la cual es modificada por la función **validar_tiro**(),que indica si el último penal pateado fue gol o no.
Una vez que se valide el penal, se le pasará el turno al próximo jugador.
Los jugadores siempre patean en un orden definido (ej: jug1, jug2, …, jug5, jug1, jug2, etc).
Existe a disposición la función **actual**() que retorna el id del pateador actual, y la función siguiente() que retorna el id del próximo pateador.

Provea una solución que sincronice los tres procesos usando solamente semáforos, asegurándose que se cumplan las reglas establecidas sin producirse deadlock ni starvation. Se deberá inicializar cada semáforo, indicando también su tipo.

10- Sincronice el siguiente código, correspondiente a un proceso que genera procesos hijos, para evitar inconsistencias, deadlocks e inanición. Además debe tener en cuenta lo siguiente:

```
int main() {
  while (true) {
    pid = fork();
    if (pid < 0) {
      log('Error');
    } else if (pid == 0) {
      log("Y yo soy el hijo");
      realizarTarea();
      // Finaliza el proceso hijo
      exit(0);
    } else { // Padre
      log(pid + " soy tu padre");
    }
  }
}
```

- El archivo donde se escriben los logs es único.
- No debe haber más de 50 procesos en ejecución
- El padre debe escribir en el log antes que el hijo recién creado.