

Sistemas Operativos

***Tp 1er Parcial***

**- Sincronización de procesos**

**Profesor:** Leandro Robles

**Comisión:** (757) 1

**Apellido y nombre:** Viltez, Hernan

**DNI:** 3089300

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++;  } |

|  |  |
| --- | --- |
| variables\_compartidas a = 1;  variables\_compartidas b = 1;  semáforo semaforo\_a = 1;  semáforo semaforo\_b = 1; |  |
| Proceso 0 | Proceso 1 |
| variable\_local d = 1;  While (TRUE){  Esperar(semáforo\_a);  a = a + d;  d = d \* d;  Liberar(semáforo\_a);  Esperar(semáforo\_b);  b = b - d;  Liberar(semáforo\_b);  } | variable\_local e = 2;  While (TRUE){  Esperar(semáforo\_b);  b = b \* e;  e = e ^ e;  Liberar(semáforo\_b);  Esperar(semáforo\_a);  a++;  Liberar(semáforo\_a);  } |

2- Dado un sistema con N procesos del mismo programa, sincronice su código mediante semáforos para respetar el límite de tres instancias del recurso usado.

|  |
| --- |
| 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 escaner 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\_escanner();  } | While (TRUE){    usar\_escanner();    usar\_impresora();  } |

|  |  |  |
| --- | --- | --- |
| N = número\_de\_procesos  Número\_de\_impresoras = 3  Número\_de\_escáneres = 2  semáforo\_impresoras = Número\_de\_impresoras  semáforo\_escáneres = Número\_de\_escáneres  semáforo\_variable\_compartida = 1 | | |
| **Proceso A** | **Proceso B** | **Proceso C** |
| While (TRUE) {  Esperar(semáforo\_impresoras);  Esperar(semáforo\_variable\_compartida);  usar\_impresora();  variable\_compartida++;  Liberar(semáforo\_variable\_compartida);  Liberar(semáforo\_impresoras);  } | While (TRUE) {  Esperar(semáforo\_variable\_compartida);  variable\_compartida++;  Liberar(semáforo\_variable\_compartida);  Esperar(semáforo\_escáneres);  usar\_escanner();  Liberar(semáforo\_escáneres);  } | While (TRUE) {  Esperar(semáforo\_escáneres); Esperar(semáforo\_impresoras);  Esperar(semáforo\_variable\_compartida);  usar\_escanner();  usar\_impresora();  variable\_compartida++;  Liberar(semáforo\_variable\_compartida);  Liberar(semáforo\_impresoras);  Liberar(semáforo\_escáneres);  } |

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

|  |  |
| --- | --- |
| semáforo semáforo\_A = 1;  semáforo semáforo\_B = 0; | |
| **Proceso A** | **Proceso B** |
| While (TRUE) {  Esperar(semáforo\_A);  Ejecutar\_A();  Liberar(semáforo\_B);  Esperar(semáforo\_A);  } | While (TRUE) {  Esperar(semáforo\_B);  Ejecutar\_B();  Liberar(semáforo\_A);  Esperar(semáforo\_B);  } |

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

(A,B,C,A,B,C...).

|  |  |  |
| --- | --- | --- |
| semáforo semáforo\_A = 1;  semáforo semáforo\_B = 0;  semáforo semáforo\_C = 0; | | |
| **Proceso A** | **Proceso B** | **Proceso C** |
| While (TRUE) {  Esperar(semáforo\_A);  Ejecutar\_A();  Liberar(semáforo\_B);  Esperar(semáforo\_A);  } | While (TRUE) {  Esperar(semáforo\_B);  Ejecutar\_B();  Liberar(semáforo\_C);  Esperar(semáforo\_B);  } | While (TRUE) {  Esperar(semáforo\_C);  Ejecutar\_C();  Liberar(semáforo\_A);  Esperar(semáforo\_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,....

|  |  |  |
| --- | --- | --- |
| semáforo semáforo\_A = 0;  semáforo semáforo\_B = 1;  semáforo semáforo\_C = 0; | | |
| **Proceso A** | **Proceso B** | **Proceso C** |
| While (TRUE) {  Esperar(semáforo\_A);  Ejecutar\_A();  Liberar(semáforo\_B);  } | While (TRUE) {  Esperar(semáforo\_B);  Ejecutar\_B();  Liberar(semáforo\_C);  } | While (TRUE) {  Esperar(semáforo\_C);  Ejecutar\_C();  Liberar(semáforo\_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.

|  |  |
| --- | --- |
|  |  |
| **KLT compilador (N instancias)** | **KLT notificador (1 instancia)** |
| While (TRUE) {  id\_programa = obtener\_nuevo\_programa();  r = compilar\_programa(id\_programa);  Esperar(mutex\_lista);  depositar\_resultado(r, lista);  Liberar(mutex\_lista);  Incrementar(resultados\_disponibles);  } | While (TRUE) {  Esperar(resultados\_disponibles);  Esperar(mutex\_lista);  r2 = retirar\_resultado(lista);  Liberar(mutex\_lista);  enviar\_email(r2);  } |

b) Asumiendo que la lista tiene un límite de M resultados como máximo.

|  |  |
| --- | --- |
| semáforo mutex\_lista = 1;  semáforo resultados\_disponibles = 0;  semáforo espacio\_en\_lista = M; | |
| **KLT compilador (N instancias)** | **KLT notificador (1 instancia)** |
| While (TRUE) {  id\_programa = obtener\_nuevo\_programa();  r = compilar\_programa(id\_programa);    Esperar(espacio\_en\_lista);  Esperar(mutex\_lista);  depositar\_resultado(r, lista);  Liberar(mutex\_lista);  Incrementar(resultados\_disponibles);  } | While (TRUE) {  Esperar(resultados\_disponibles);    Esperar(mutex\_lista);  r2 = retirar\_resultado(lista);  Liberar(mutex\_lista);    Incrementar(espacio\_en\_lista);  enviar\_email(r2);  } |

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 function log() imprime por pantalla el valor actual de pistas libres.