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.