SISTEMAS OPERATIVOS. PED I. (2014/2015)

Alumno: Oscar Rubio Fauria

DNI: 43749284H

Centro Asociado: Cadiz

Ejercicio 1.

I) **FALSO.** (seccion 4.5.3 del libro base, pg 148)

Un monitor garantiza implicitamente la exclusion mutua sobre recursos compartidos por varios procesos concurrentes. Simplemente hay que definir dentro del monitor los recursos compartidos.

Por otra parte, un monitor si se define correctamente garantiza la sincronizacion de procesos mediante el uso de variables de condicion y de las operaciones wait_mon y signal_mon, ya que el protocolo de sincronizacion se implementa dentro del propio monitor.

Los monitores no permiten realizar la comunicación entre procesos porque no posibilitan el intercambio explicito de informacion entre ellos.

II) **VERDADERO.** (seccion 2.3.4 del libro base, pg 65)

El tiempo de commutacion es el tiempo que se necesita para realizar la operación de cambio de proceso. La magnitud de este tiempo depende principalmente de factores asociados al hardware del computador como por ejemplo, la velocidad de lectura/escritura de la memoria principal, el numero de registros del procesador cuyo contexto hay que salvar o cargar, la velocidad del procesador y la existencia de instrucciones especiales en el repertorio del procesador para salvar o cargar todos los registros.

III) **VERDADERO.** (seccion 3.6.2 del libro base, pg 92)

El algoritmo de planificación SJF minimiza el tiempo de espera medio de un conjunto dado de procesos. A diferencia del algoritmo FCFS, el algoritmo SJF favorece a los procesos cortos frente a los procesos largos, el tiempo de espera de los procesos cortos disminuye mas de lo que aumenta el tiempo de espera de los procesos largos. Por tanto, el tiempo de espera promedio se decrementa.

IV) **VERDADERO.** (seccion 1.5 del libro base, pg 41)

Los programas usuario se ejecutan en modo usuario y su acceso al hardware esta restringido. Para poder usar los recursos hardware, los programas usuario deben solicitar su uso al sistema operativo mediante la realizacion de llamadas al sistema, que conforman la interfaz entre los programas de usuario y el nucleo.

Cuando un programa de usuario invoca a una llamada al sistema se produce una trampa que provoca la commutacion hardware de modo usuario a modo supervisor y transfiere el control al nucleo. El nucleo examina el identificador numerico de la llamada y los parametros de la llamada para poder localizar y ejecutar a la rutina del nucleo asociada a la llamada al sistema. Cuando dicha rutina finaliza, almacena el resultado en algun registro y provoca la commutacion hardware de modo supervisor a

modo usuario para que el proceso de usuario que invoco la llamada continue su ejecucion.

Ejercicio 2.

Lo resolvemos mediante el uso de la pagina 91 del libro base

Tr = Tiempo de retorno o estancia

Ts = Tiempo de servicio o ejecucion

Te = Tiempo de espera

Te = Tr – Ts

Tr = Te + Ts

Tr medio = (Tr1 + Tr2 + Tr3) / 3 = 20

Trabajo	**Ts**	**Te**
T1	X+5	5
T2	X+13	5
T3	X+4	X+4

Tr1 = X+5+5 = X+10

Tr2 = X+13+5 = X+18

Tr3 = X+4+X+4 = 2X+8

Tr medio = ((2X+8)+(X+10)+(X+18))/3 = 20

4X+36 = 60

X = (60-36)/4 = 24/ 4 = **6 ut**

Solucion: La duracion de las rafagas x de CPU es de 6 unidades de tiempo

Ejercicio 3.

MONITORES. (apartado 4.5 del libro base pags. 145 -150)

Codigo de inicializacion de un monitor.

monitor [Nombre del monitor]

/* Declaracion de variables locales */

/* Declaracion de las variables de condicion */

/* Declaracion de lor procedimientos del monitor */

/* Inicializacion del monitor */

end monitor

Las **variables de condicion** son las herraminetas que se utilizan para la sincronizacion de procesos en un monitor.

Sobre las variables de condicion se pueden realizar dos operaciones:

- **wait_mon(condicion).** El proceso dentro del monitor del monitor que realiza esta operación queda suspendido en una cola de procesos asociada al cumplimineto de la condicion. Esta operación siempre produce el bloqueo del proceso que le invoca.
- **signal_mon(condicion).** Comprueba si la cola de procesos asociada a la condicion contiene algun proceso bloqueado. En caso afirmativo se desbloquea a un proceso.
  En el ejercicion se pide utilizar la **solucion de Hansen** para el proceso que invoca esta operación. El proceso invocador sale del monitor inmediatamente. Luego esta operación siempre apareceria como la sentencia final del procedimiento de un monitor.

Codigo solucion del ejercicio:

monitor jaula

*/*declaracion variables locales*/*

boolean columpio_libre; *//informa del estado del columpio para el proceso canario que*

//acceder

int contador; *//se encarga de controlar el numero de platos ocupados por procesos*

*/*declaracion de las variables de condicion*/*

condicion cola_comer, cola_columpio; *//controlan el acceso de los procesos a los platos*

//y al columpio

*/*declaracion de procedimientos*/*

void cojer_plato()

{

if(contador==3) wait_mon(cola_comer); *//si el contador es igual a 3 quiere decir*

//que todos los platos estan ocupados por tanto el proceso se bloquea y se traslada

```
        //a la cola de procesos de la condicion cola_comer a la espera de un signal_mon
        //de la condicion que lo desbloquee
        contador = contador + 1; //se incrementa en una unidad porque el proceso accede
        //al plato
        //no hay una sentencia final signal_mon porque el proceso aun tiene que realizar
        //la operación de comer y por tanto no se puede liberar ningun proceso
}
void dejar_plato()
{
        contador = contador – 1; //como el proceso abandona el plato tenemos que
                                 //decrementar en una unidad el contador
        signal_mon(cola_comer); //permitira el acceso de algun proceso que este
                                 //bloqueado en la cola
}
//misma idea que con los platos pero en vez de un contador, al existir solo un columpio
//utilizamo una variable booleana que en nos indica si el columpio esta libre.
void cojer_columpio()
{
        if(!columpio_libre) wait_mon(cola_columpio);
        columpio_libre = false;
}
void dejar_columpio()
{
        columpio_libre = true;
        signal_mon(cola_columpio);
}
```

```
        /*inicializacion*/
        {
                contador = 0; //indica que los 3 platos no estan ocupados
                columpio_libre = true; //indica que el columpio no esta ocupado
        }
end monitor
void canario()
{
        jaula.cojer_plato(); //primero se intenta acceder al plato como indica el enunciado
        comer(); //operación interna del procedimiento canario que ejecuta la accion de comer
                 //no se implementa porque no es lo que pide el ejercicio
        jaula.dejar_plato(); //cuando se realiza este procedimiento del monitor indica que el
                             //canario ha comido por tanto ya puede intentar acceder al columpio
        jaula.cojer_columpio(); //el proceso intenta acceder al columpio
        columpiar(); //realiza la accion de columpiarse
        jaula.dejar_columpio(); //el proceso deja el columpio y finaliza su ejecucion
}
void main()
{
        ejecucion_concurrente(canario, … , canario);
}
```