SISTEMAS OPERATIVOS

PRIMERA PRUEBA

DE

EVALUACIÓN A DISTANCIA

(PED1)

Curso 2013-2014

José Manuel Sáez Vicioso

DNI: 75765391 – X

Centro asociado de Cádiz

SISTEMAS OPERATIVOS
Primera Prueba de Evaluación a Distancia (PED1)

1. Explique **razonadamente** si las siguientes afirmaciones son verdaderas o falsas:

I) *(1 p)* Se denomina *sobrecarga del sistema* al número de procesos que se encuentran en el estado bloqueado en espera de que se produzca un determinado evento.
II) *(1 p)* El *planificador a largo plazo* de un sistema operativo da más prioridad a los trabajos limitados por CPU que a los trabajos limitados por E/S.
III) *(1 p)* El micronúcleo de un sistema operativo con estructura extensible se encarga, entre otras tareas, de la gestión de la memoria virtual.

Solución

I) **Falso**. La sobrecarga del sistema es el tiempo que el procesador se encuentra ocupado ejecutando código del sistema operativo asociado a tareas y servicios de administración que no se pueden contabilizar a ningún proceso en particular.

II) **Falso.** El planificadora largo plazo debe intentar seleccionar trabajos de tal forma que en el sistema exista una mezcla adecuada de procesos limitados por CPU y procesos limitados por E/S.

III) **Falso.** El micronúcleo se encarga de realizar únicamente los servicios absolutamente esenciales del sistema operativo, aquellos que dependen de la arquitectura de la máquina y que son independientes del tipo de sistema operativo.

La gestión de memoria virtual es independiente de la arquitectura de la máquina y depende del sistema operativo, ésta se implementa como extensión del núcleo y se ejecuta en modo usuario.

2. Una sala de exposiciones tiene una capacidad para V visitantes y dispone de un guía. Si la sala está llena los visitantes deben esperar en una cola para entrar. Una vez dentro de la sala se distinguen dos tipos de visitantes: los visitantes guiados y los visitantes libres. Un visitante guiado cuando entra en la sala se dirige a un punto de encuentro existente donde espera junto con otros visitantes a que el guía les avise para iniciar la visita guiada. El visitante libre realiza la visita por su cuenta. Ambos tipos de visitantes cuando terminan la visita abandonan la sala. El guía si no hay nadie en el punto de encuentro descansa sentado en una silla, solo cuando hay 10 visitantes en el punto de encuentro se levanta y avisa a los visitantes allí presentes para que le sigan. Cuando termina una visita guiada vuelve a su silla y descansa al menos 7 minutos antes de iniciar otra visita guiada. Escribir el pseudocódigo de un programa que coordine la actividad de los visitantes y del guía en la sala de exposiciones usando:

a) *(1 p)* Semáforos generales.
b) *(1.5 p)* Semáforos binarios.
c) *(1.5 p)* Paso de mensajes, suponer que la comunicación es indirecta a través de buzones y que se dispone de la operación *send* sin bloqueo y de la operación *receive* con bloqueo.

El pseudocódigo del programa que se realice en cada apartado debe tener cinco partes: declaración de variables, código visitante guiado, código visitante libre, código del guía, y código para inicializar los elementos de sincronización (semáforos o cola de mensajes) y lanzar la ejecución concurrente de los procesos.

a) Semáforos generales.

```
/* Declaración variables y semáforos */
#define TRUE 1
semáforo S1, S2, S3;
int cola_guiada=0, i=0, capacidad = V;

void visitante_libre()  /* Proceso visitante libre */
{
        wait_sem(S1);
        ver_exposición();
        signal_sem(S1);
        abandonar_sala();       /* este paso se podría omitir */
}

void visitante_guiado()         /* Proceso visitante guiado */
{
        wait_sem(S1);
        ir_a_punto_de_encuentro();
        wait_sem(S2);           /* espera a que el guía lo avise */
                                /* ¿necesito un contador? */
        seguir_guía();          /* sigue al guia */
        signal_sem(S1);         /* se va */
        abandonar_sala();
}
```

```
void guia()             /* Proceso guía */
{
        while(TRUE)
        {
        /* para el guia necesitaría un contador y por tanto un semáforo binario de
        exclusión mutua y un bucle para avisar a todos los que están en la cola */
                        for(i=0;i<cola;i++) signal_sem(S2);

                        guía_a_los_visitantes();
                        descansa_7_minutos();
                }
        }
}

void main()
{
        init_sem(S1, capacidad);
        init_sem(S2, 0);

        ejecución_concunrrente(visitante_libre, …, visitante_libre,
        visitante_guiado, …, visitante_guiado, guia);
```

b) Semáforos binarios.

Variables globales y semáforos:

- *cola_entrada*: Variable global tipo entero para llevar la cuenta del número de visitantes dentro de la sala de exposiciones.
- *cola_guiada*: Variable global tipo entero para llevar la cuenta del número de visitantes en el punto de encuentro.
- *S1*. Semáforo binario que se utiliza para garantizar la exclusión mutua en el uso de la variable global *cola_entrada*.
- *S2*. Semáforo binario que se utiliza para garantizar la exclusión mutua en el uso de la variable global *cola_guiada*.
- *S3*. Semáforo binario que se utiliza para sincronizar la entrada al salón de exposiciones.
- *S4*. Semáforo binario que se utiliza para implementar la espera de los clientes a acceder a la visita guiada.

```
/* Declaración variables y semáforos */
#define TRUE 1
#define capacidad V

semáforo_binario S1, S2, S3, S4;
int cola_entrada=0, cola_guiada=0, i=0;

void visitante_libre()  /* Proceso visitante libre */
{
        wait_sem(S1);
        cola_entrada = cola_entrada + 1;
        if (cola_entrada > capacidad)
        {
                signal_sem(S1);
                wait_sem(S3);           /* espera en la cola de entrada */
        }
        else
        {
                signal_sem(S1);
                ver_exposición();
                signal_sem(S3);
        }
        /* Salir */
        wait_sem(S1);
        cola_entrada = cola_entrada - 1;
        signal_sem(S1);
        abandonar_sala();
}
```

```
void visitante_guiado()          /* Proceso visitante guiado */
{
        wait_sem(S1);
        cola_entrada = cola_entrada + 1;
        if (cola_entrada > capacidad)
        {
                signal_sem(S1);
                wait_sem(S3);           /* espera en la cola de entrada */
        }
        else signal_sem(S1);

        ir_a_punto_de_encuentro();
        signal_sem(S3);

        wait_sem(S2);
        cola_guiada = cola_guiada + 1;
        signal_sem(S2);

        wait_sem(S4);           /* espera a que el guía lo avise*/
        seguir_guía();

        /* Salir */
        wait_sem(S1);
        cola_entrada = cola_entrada - 1;
        signal_sem(S1);
        abandonar_sala();
}
```

```
void guia()             /* Proceso guía */
{
        while(TRUE)
        {
                wait_sem(S2);
                if( cola_guiada < 10 )
                {
                        signal_sem(S2);
                        descansa_en _la_silla();
                }
                else
                {

        /* Avisa a todos los visitantes que están en el punto de encuentro*/
                        for ( i=0; i < cola_guiada; i++)  signal_sem(S4);

                        /* No hay nadie en el punto de encuentro */
                        cola_guiada = 0;
                        signal_sem(S2);

                        guía_a_los_visitantes();
                        descansa_7_minutos();
                }
        }
}

void main() /* Inicialización de semáforos y ejecución concurrente */
{
        init_sem(S1, 1);
        init_sem(S2, 1);
        init_sem(S3, 0);
        init_sem(S4, 1);
        ejecución_concunrrente(visitante_libre, …, visitante_libre,
        visitante_guiado, …, visitante_guiado, guia);
}
```

c) Paso de mensajes.

3. (3 p) Se dispone del siguiente conjunto de trabajos de usuarios para su planificación:

Trabajo	Tiempo de llegada (ut)	Duración de las ráfagas (ut)
T0	0	3, 2, 2, 2, 1
T1	1	2, 3, 1, 2
T2	3	8, 1, 1

El sistema operativo utiliza un algoritmo de planificación por prioridades de tipo no expropiativo. La prioridad más alta corresponde al valor 0 y la más baja al valor 169. Cuando un trabajo de usuario entra en el estado preparado para ejecución recibe una prioridad igual a 100. La prioridad de todos los trabajos de usuarios se recalcula cada vez que se termina de ejecutar una ráfaga de CPU de un trabajo. Si se trata del trabajo que acaba de usar la CPU entonces su prioridad se recalcula de la siguiente forma:

Nueva prioridad = min{prioridad actual + duración de la última ráfaga, 169}

Para el resto de trabajos sus prioridades se recalculan de la siguiente forma:

Nueva prioridad = max{0, prioridad actual - 1}

A la hora de planificar un trabajo el sistema selecciona para planificar al que tiene la prioridad más alta (valor numérico de la prioridad más pequeño). Si existen varios trabajos con la misma prioridad entonces escoge aquel trabajo que lleve acumulado un menor tiempo uso de CPU. En caso de que existan varios trabajos con la misma prioridad y con el mismo tiempo de uso de CPU acumulado entonces selecciona para planificar al trabajo que haya llegado antes al sistema. Representar el diagrama de uso de la CPU y calcular el tiempo de retorno y el tiempo de espera de cada trabajo.

Solución

No he tenido tiempo para hacerlo.