SISTEMAS OPERATIVOS

Primera Prueba de Evaluación a Distancia (PED1)

Nombre: Rafael Canca Repiso
DNI: 75790634-E
Centro asociado: Las Palmas de Gran Canaria

Ejercicio 1

I) Esta afirmación es Falsa, el hecho que le demos a un proceso solamente aquellas peticiones de recursos que garanticen que no conducirán a un estado de interbloqueo no es suficiente para prevenir el interbloqueo según la estrategia de prevención de interbloqueos. La estrategia de prevención de interbloqueos consiste en eliminar la aparición de alguna de las cuatro condiciones necesarias y suficientes para que se produzca un interbloqueo: exclusión mutua, retención y espera, no existencia de expropiación y espera circular.

En el caso que estamos hablando, por ejemplo, se podría dar la expropiación.

II) La planificación expropiativa produce un menor sobrecarga al sistema que una planificación no expropiativa.

Esta afirmación es Falsa, justamente es lo contrario, es decir, la planificación expropiativa produce una mayor sobrecarga al sistema que una no expropiativa, esto es debido a que se ejecuta con mayor frecuencia el planificador y se realizan más cambios de proceso.

III) Esta afirmación es Verdadera por tres razones. La primera es que crear un hilo nuevo dentro de un proceso ya existente requiere de menos tiempo que la creación de un nuevo proceso. La segunda es que el tiempo para finalizar un hilo es menor que el tiempo necesario para finalizar un proceso. Y la tercera es que un cambio de proceso requiere más tiempo que un cambio de hilo dentro de un mismo proceso.

IV) Esta afirmación es Falsa. No necesariamente los sistemas multiacceso requieren multiprogramación. Podemos ver el caso de los sistemas informáticos dedicados al procesamiento de transacciones que soportan centenares de terminales activos que son atendidos por un único programa.

Ejercicio 2.

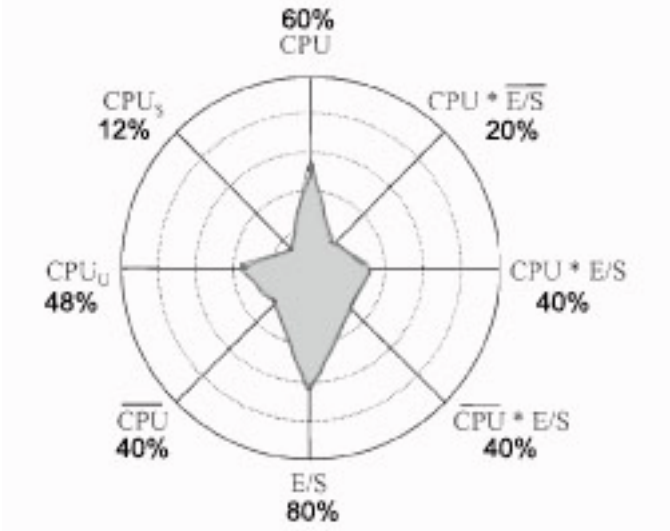
Si tan sólo nos fijásemos en el grafo veríamos que existe un ciclo y podríamos deducir que se produciría un interbloqueo. Pero al leer el enunciado observamos que el sistema que estamos evaluando permite expropiación en sus recursos, y por lo tanto estamos impidiendo que se de una de las cuatro condiciones necesarias para que se produzca el interbloqueo.

Ejercicio 3.

Calculamos los índices de prestaciones del diagrama de Kiviat-Kent a partir del diagrama de uso.

- La CPU ha sido utilizada durante 35ut +25ut=60 ut, luego como el tiempo total de observación es de 100 ut tenemos que CPU=60%.
- La CPU ha estado en modo supervisor ejecutando código del sistema operativo tal y como indica el enunciado CPU_S=12%
- El tiempo que la CPU ha estado en modo usuario ejecutando código de procesos de usuario lo obtenemos de restar el tiempo que ha estado en uso la CPU y el tiempo que ha estado ejecutándose en modo supervisor, es decir 60-12, es decir CPU_U=48%
- El tiempo de inactividad de la CPU es $\overline{CPU}$= 100-60=40%
- La CPU y los dispositivos de E/S han estado en uso simultáneamente durante CPU-E/S=20+20=40%
- La CPU ha estado en uso sin estarlo los dispositivos de E/S durante CPU $\overline{E/S}$=15+5=20%
- Los dispositivos de E/S han estado en uso sin estarlo la CPU durante $\overline{CPU}$ – E/S=15+25=40%
- Los dispositivos de E/S han estado en uso durante E/S = 35+45=80%

Con estos porcentajes ya podemos representar el diagrama de Kiviat-Kent



Podemos ver como este diagrama nos indica que el sistema está limitado por el subsistema de E/S (80%) aunque vemos que presenta un buena utilización de la CPU en modo usuario y en modo supervisor.

Ejercicio 4. Programa coordinación

Podemos tratar este caso, prácticamente igual que si fueran semáforos generales. Tan sólo tendríamos que declarar los semáforos como tipos binarios.

```
/*declaración de variables*/
semaforo_binario S1,S2;

/*código del proceso A*/
void procesoA()
{
        tareas_1();
        signal_sem(S2);
        wait_sem(S1)
        tareas_2();

}

/*código del proceso B*/
void procesoB()
{
        tareas_3();
        signal_sem(S1);
        wait_sem(S2)
        tareas_4();

}

/*Inicialización de semáforos y ejecución concurrente*/
void main()
{
        init_sem(S1,0);
        init_sem(S2,0);
        ejecucion_concurrente(proceso_A,proceso_b);
}
```