Centro de Investigación en Computo

Tarea 1. Investigación de cálculo de quantum en sistemas operativos

Materia: Sistemas Operativos

27 de septiembre de 2025

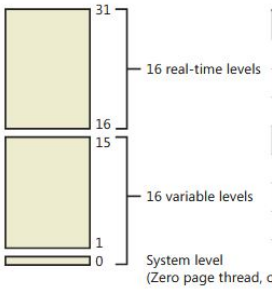
1. Objetivo

Realizar la investigación de cómo se realiza el cálculo del parámetro quantum en los sistemas operativos de Windows y Linux o Unix.

2. Quantum en Windows

2.1. Conceptos previos

Windows implementa un sistema de programación de tareas de tipo *preemptive*, el cual está basado en prioridades. Cada proceso puede tener una prioridad desde 0 (prioridad más baja) a 31 (prioridad más alta) divididos en 6 diferentes clases: *Realtime, high, Above normal, Normal, Below normal e Idle*. Las prioridades son asignadas de dos formas diferentes , a través del API (Interfaz de Programación de Aplicaciones) de Windows o el kernel de Windows.



Priority Class Relative Priority	Realtime	High	Above Normal	Normal	Below Normal	Idle
Time Critical (+ SATURATION)	31	15	15	15	15	15
Highest (+2)	26	15	12	10	8	6
Above Normal (+1)	25	14	11	9	7	5
Normal (0)	24	13	10	8	6	4
Below Normal (-1)	23	12	9	7	5	3
Lowest (-2)	22	11	8	6	4	2
Idle (- SATURATION)	16	1	1	1	1	1

Figura 1: Prioridad de procesos y asignación de prioridad por el kernel.

Adicionalmente, se cuenta con la afinidad del procesador, la cual determina en cuál procesador se tiene programada la tarea o proceso para su ejecución.

El encargado de realizar la programación de tareas es llamado el despachador de base de datos; este elemento realiza el seguimiento de los procesos que están listos para ser ejecutados y es implementado por colas para cada procesador disponible.

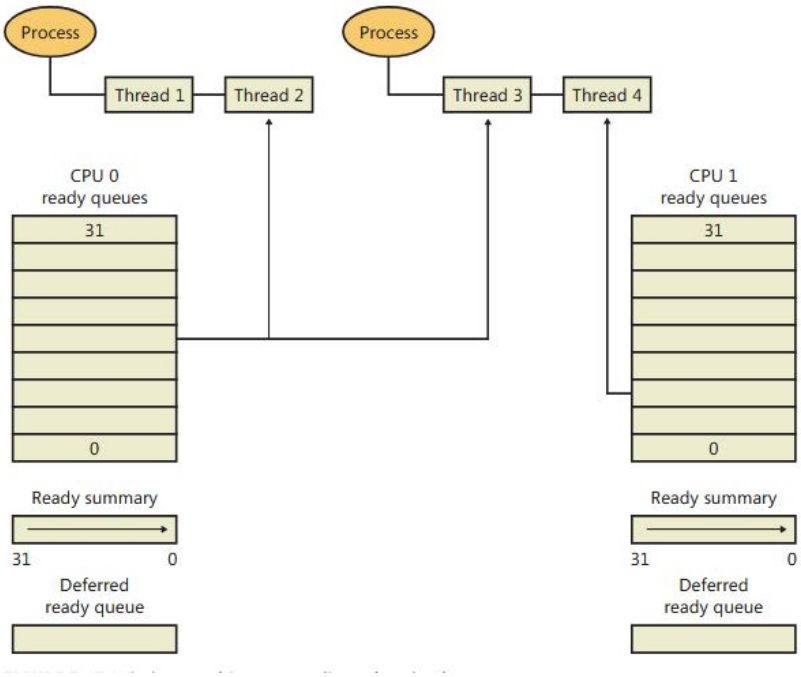


Figura 2: Despachador de base de datos.

2.2. Calculo de quantum

Para Windows en las versiones de cliente, los procesos tienen un quantum de 2 intervalos de reloj y para las versiones de servidor se tiene un intervalo de 12 intervalos de reloj. Este valor cambia acorde a las características del equipo y su *hardware*, cada que el sistema inicial se calcula el número de ciclos a los cuales equivale un quantum, dicho valor es guardado en la variable *KiCyclesPerClockQuantum*, este es calculado multiplicando el valor de la velocidad del procesador en Hz con los segundos del intervalo del reloj almacenados en la variable *KeMaximumIncrement*.

The Windows *GetSystemTimeAdjustment* function returns the clock interval. To determine the clock interval, download and run the Clockres program from Windows Sysinternals (www.microsoft.com/technet/sysinternals). Here's the output from a dual-core 64-bit Windows 7 system:

```
C:\>clockres

ClockRes v2.0 - View the system clock resolution
Copyright (C) 2009 Mark Russinovich
SysInternals - www.sysinternals.com

Maximum timer interval: 15.600 ms
Minimum timer interval: 0.500 ms
Current timer interval: 15.600 ms
```

Figura 3: Obtención del intervalo del reloj a través de utileria clockres

3. Quantum en Linux

En los kernel modernos de Linux, el planificador de procesos por defecto es CFS *(Completely Fair Scheduler)* y la configuración de cómo se calcula el quantum se encuentra en el archivo fair.c ubicado en la ruta /kernel/sched. En este archivo se puede observar la función llamada *sched_slice*.

```
static u64 sched_slice(struct cfs_rq *cfs_rq, struct sched_entity *se)
{
        unsigned int nr_running = cfs_rq->nr_running;
        struct sched_entity *init_se = se;
        unsigned int min_gran;
        u64 slice;

        if (sched_feat(ALT_PERIOD))
                nr_running = rq_of(cfs_rq)->cfs.h_nr_running;

        slice = __sched_period(nr_running + !se->on_rq);

        for_each_sched_entity(se) {
                struct load_weight *load;
                struct load_weight lw;
                struct cfs_rq *qcfs_rq;

                qcfs_rq = cfs_rq_of(se);
                load = &qcfs_rq->load;

                if (unlikely(!se->on_rq)) {
                        lw = qcfs_rq->load;

                        update_load_add(&lw, se->load.weight);
                        load = &lw;
                }
                slice = __calc_delta(slice, se->load.weight, load);
        }

        if (sched_feat(BASE_SLICE)) {
                if (se_is_idle(init_se) && !sched_idle_cfs_rq(cfs_rq))
                        min_gran = sysctl_sched_idle_min_granularity;
                else
                        min_gran = sysctl_sched_min_granularity;

                slice = max_t(u64, slice, min_gran);
        }

        return slice;
}
```

Figura 4: Función sched_slice en archivo fair.c

La función recibe la cola de CFS(cfs_rq) y el grupo de procesos (se) y devuelve el quantum en nanosegundos. Para esto último se revisa la cantidad de procesos y si estos son pocos se devuelve un tiempo por defecto; en otro caso se calcula la proporcionalidad a través de la siguiente fórmula y se calcula una granularidad mínima.

$$\text{slice}(se) = \frac{\text{periodo}}{\text{peso total}} \times \text{peso}(se) \tag{1}$$

Nota: Fórmula del quantum en Linux CFS.

Referencias

[1] Ingo Molnar. (2007). *Completely Fair Scheduler*. Disponible en: `https://www.kernel.org/doc/Documentation/scheduler/sched-design-CFS.txt`

[2] Russinovich, M., Solomon, D., Ionescu, A. (2012). *Windows Internals, Part 1 (6th Edition)*. Microsoft Press.

[3] Linux Kernel Organization. (2025). *kernel/sched/fair.c, línea 773*. Disponible en: `https://elixir.bootlin.com/linux/v6.0.19/source/kernel/sched/fair.c#L773`

[4] Microsoft. (2021). *CPU Analysis*. Disponible en: `https://learn.microsoft.com/en-us/windows-hardware/test/wpt/cpu-analysis`

[5] Microsoft. (2025). *Scheduling Priorities*. Disponible en: `https://learn.microsoft.com/en-us/windows/win32/procthread/scheduling-priorities`