Universidad del Valle de Guatemala

Programación orientada a objetos

Semestre – 2

Inge. Erick Marroquín

Estudiante Nathalie Elías – 251290

Tipos de procesos en sistemas operativos

En el contexto de los sistemas operativos (SO), los procesos representan unidades fundamentales de ejecución que permiten la multitarea y la gestión eficiente de recursos computacionales. Esta investigación examina los tres tipos principales de procesos mencionados en la situación del ejercicio: procesos de CPU (CPU-bound), procesos de entrada/salida (I/O-bound) y deamons (procesos en segundo plano). El análisis se basa en principios establecidos de la arquitectura de SO, con énfasis en sus características operativas, impactos en el rendimiento y ejemplos de aplicación en entornos reales.

Procesos de CPU (CPU-bound)

Los procesos de CPU, también denominados CPU-intensivos o CPU-bound, se definen como aquellos que requieren un consumo sostenido y elevado de ciclos de procesamiento de la unidad central de procesamiento (CPU) para ejecutar tareas computacionales complejas. Estos procesos exhiben ráfagas prolongadas de ejecución activa, con interrupciones mínimas derivadas de operaciones de entrada/salida, lo que los hace particularmente sensibles a la capacidad de cómputo disponible, incluyendo el número de núcleos y la frecuencia de reloj. En términos de gestión de recursos, su ejecución puede llevar a un alto nivel de utilización de la CPU, potencialmente causando contención en sistemas multi-tarea si no se aplican mecanismos de sheduling apropiados, como el Round-Robin o colas de retroalimentación multinivel.

En aplicaciones del mundo real, los procesos de CPU son prevalentes en escenarios que demandan cálculos algorítmicos intensivos. Un ejemplo es la compilación de código fuente mediante el GNU Compiler Collectioo (GCC) en entornos Linux, donde el análisis sintáctico, la optimización de código intermedio y la generación de binarios consumen hasta el 100% de la CPU durante fases críticas, especialmente en compilaciones paralelas con herramientas como “make -jN” (Silberschatz et al., 2018). Otro caso es el renderizado gráfico en software como Blender, que emplea algoritmos de trazado de rayos para procesar escenas 3D complejas, saturando múltiples núcleos en estaciones de trabajo de alto rendimiento para producir animaciones o simulaciones visuales. Adicionalmente, en el ámbito del aprendizaje automático, frameworks como TensorFllow ejecutan interacciones de entrenamiento de modelos neuronales, realizando operaciones matriciales masivas que dependen exclusivamente del poder de cómputo de la CPU para convergencia.

Procesos de Entrada/Salida (I/O-Bound)

Los procesos de entrada/salida, conocidos como I/O-bound, se caracterizan por una interacción frecuente y dominante con dispositivos periféricos externos, tales como discos duros, redes o interfaces de usuario. A diferencia de los procesos de CPU, estos alternan ráfagas cortas de ejecución en la CPU con periodos extendidos de bloqueo o espera, durante los cuales el proceso cede el control de la CPU al scheduler para permitir la multitarea eficiente. El rendimiento de estos procesos está limitado principalmente por la latencia y el ancho de banda de los dispositivos I/O, en lugar del poder cómputo, lo que los hace ideales para algoritmos de planificación que priorizan la responsividad , como el Shortest Remainin Time Fistr.

En el ámbito práctico, los procesos I/O.bound son esenciales en operaciones de transferencia de datos y servicios de red. Por instancia, el comando “cp” (copy) en sistemas Unix-like realiza copias de archivos entre discos HDD/SSD o unidades de almacenamiento de E/S del kernel maneja la lectura y escritura de bloques, resultando en un uso bajo de CPU pero alto tiempo de espera (Tanenbaum & Bos, 2022). Un ejemplo en servidores web es el proceso manejado por Apache HTTP Web Services (AMS), procesando solicitudes HTTP en bursts breves intercalados con pausas por la latencia de red, lo que puede representar hasta el 90% del ciclo de vida del proceso. Asimismo, los drivers de periféricos, como los utilizados en impresoras mediante el sistema CUPS en Linux, esperan confirmaciones de hardware USB o paralelas, integrando interrupciones del kernel para sincronizar la transferencia de buffers de datos.

Daemons (Procesos en Segundo Plano)

Los daemons, o procesos residentes en segundo plano, constituyen una clase especializada de procesos no interactivos que operan de manera autónoma y persistente, proporcionando servicios esenciales al sistema operativo y a las aplicaciones sin requerir intervención directa del usuario. Estos procesos se inician típicamente durante el arranque del SO (por ejemplo, a través de scripts init o gestores como systemd en Linux), ejecutándose en modo detached con un consumo moderado de CPU y respondiendo a eventos asincrónicos mediante señales (signals) o temporizadores. Su diseño enfatiza la disponibilidad continua y la integración con mecanismos de comunicación inter-proceso (IPC), como pipes o sockets, para mantener la estabilidad del sistema sin interferir en el foreground.

Ejemplos representativos en sistemas operativos reales incluyen el daemon SSH (sshd), que en distribuciones Linux como Ubuntu monitorea el puerto 22 para autenticar y gestionar conexiones remotas seguras, escalando dinámicamente para múltiples sesiones sin una interfaz gráfica visible (Bach, 1986). Otro caso es el daemon de planificación de tareas (crond), responsable de ejecutar comandos programados según el archivo /etc/crontab, como actualizaciones automáticas de software o limpiezas de logs a intervalos específicos, operando en background para garantizar la automatización sin carga adicional al usuario. Finalmente, el daemon de registro de eventos (rsyslogd) captura y archiva mensajes del kernel y servicios (por ejemplo, errores de hardware o accesos de seguridad) en directorios como /var/log, implementando rotación de archivos y notificaciones opcionales por correo electrónico para fines de auditoría y diagnóstico.

Esta categoría subraya la importancia de la persistencia en el diseño del simulador, donde los daemons se representan mediante ejecuciones cíclicas controladas, integrándose polimórficamente en la lista general de procesos para simular servicios de soporte.

Referencias

Bach, M. J. (1986). *The design of the UNIX operating system*. Prentice-Hall. (Edición original; conceptos actualizados en implementaciones modernas como Linux).

Silberschatz, A., Galvin, P. B., & Gagne, G. (2018). *Operating system concepts* (10ª ed.). John Wiley & Sons. (Capítulo 5: Process Scheduling).

Tanenbaum, A. S., & Bos, H. (2022). *Modern operating systems* (5ª ed.). Pearson. (Capítulo 2: Processes and Threads).