CC4302
Sistemas Operativos
Profesor: Luis Mateu

- Resumen de conceptos
- Estrategias de scheduling
- FCFS: First Come First Served
- Shortest Job First
- Prioridades
- Round Robin

Resumen

- Procesos livianos (threads) vs procesos pesados (procesos Unix)
- Preemptiveness
- Scheduling de procesos y el scheduler de procesos
- Interrupciones y el timer
- Estados de un proceso
- El descriptor de proceso
- Cambio de contexto
- Colas de scheduling
- Cambios de contexto implícito vs. cambio de contexto ***explícito*** (cuando el cambio de contexto ocurre producto de una acción del mismo proceso, y no producto de una interrupción del timer o del disco)
- El identificador de proceso (pid)
- Ráfagas de CPU

Estrategias de scheduling

- Se busca privilegiar alguna variable estadística: el uso de CPU, tiempo de despacho o tiempo de respuesta
- Al principio: maximizar el uso de la CPU (porque era costosa), y también busca simplicidad, igualdad
- Estrategia: FCFS o *First Come First Served*
- Se atienden las ráfagas de los procesos por orden de llegada
- Es non preemptive
- Ejercicio: las siguientes son las ráfagas de CPU de los procesos P1, P2 y P3. Entre paréntesis se coloca la duración del estado de espera.
  - P1: 3 (4) 3 (3) 2
  - P2: (1) 2 (3) 4 (4) 2
  - P3: (4) 4 (2) 4
- Graficar el uso de CPU para FCFS en un monoprocesador

Los problemas de FCFS

- No sirve para sistemas interactivos
- Mal tiempo de despacho promedio
- Es el tiempo que transcurre desde que llega una ráfaga (instante en que el proceso pasa de estado de espera a READY o RUN) hasta que termina de atenderse (instante en que el proceso pasa nuevamente a estado de espera)
- Ejercicio: graficar
  - P0: 10
  - P1: (5) 10
  - P2: (6) 5
  - P3: (7) 5

  vs.

  - P0: 10
  - P1: (7) 10
  - P2: (5) 5
  - P3: (6) 5
- En el primer ejemplo el tiempo de despacho promedio es 19, mientras que en el segundo es 15.6
- Se puede demostrar que atendiendo primero las ráfagas más cortas se minimiza el tiempo de despacho promedio
- Además, FCFS tiende a atender primero los procesos intensivos en CPU, ocupando el 100% de CPU, pero dejando para el final los procesos intensivos en E/S, desperdiciando CPU
- Mejor sería mezclar la atención de ambos tipos de procesos

SJF: Shortest Job First

- Se atiende primero la ráfaga más corta, es decir se cede la CPU a aquel proceso cuya próxima ráfaga será la más corta entre todos los procesos READY
- ¿Cómo conocer cuál será la más corta?
- El pasado es un buen predictor del futuro
- Considerando que ya se han ejecutado *n* ráfagas, se estima la duración de la ráfaga *n+1* del proceso P con esta fórmula: $\tau_{n+1}^{P} = \alpha t_n^p + (1-\alpha)\tau_n^P$ en donde:
  - τ_{n+1}^{P} es el predictor de la próxima ráfaga
  - t_n^p es la duración efectiva de la ráfaga n de P
  - τ_n^P es el predictor de la ráfaga n de P
  - α ponderador para la última ráfaga
- Típicamente $\alpha = 0.5$
- La fórmula es equivalente a:

$$\tau_{n+1}^{P} = \alpha t_n^p + (1-\alpha)\alpha t_{n-1}^{P} + (1-\alpha)^2 \alpha t_{n-2}^{P} + (1-\alpha)^3 \alpha t_{n-3} + \cdots$$

- La primera fórmula es más rápida de calcular

Variantes de SJF

- Por simplicidad puede ser non preemptive
- Pero también puede ser preemptive:
  - Si la duración de la ráfaga supera el siguiente mejor predictor, se le quita la CPU
  - Si aparece una ráfaga con mejor predictor, se le quita la CPU
  - Etc.
- Desventaja: hambruna para los procesos intensivos en CPU
- Ejercicio: las siguientes son las ráfagas de CPU de los procesos P1, P2 y P3 del primer ejercicio.
  - P1: 3 (4) 3 (3) 2
  - P2: (1) 2 (3) 4 (4) 2
  - P3: (4) 4 (2) 4
- Graficar el uso de CPU para SJF en un monoprocesador considerando $\alpha = 1$ non preemptive

Prioridades

- A cada proceso se asigna una prioridad, según su importancia
- Se cede la CPU al proceso que tiene la mejor prioridad
- Las prioridades pueden ser estáticas o dinámicas
- SJF es un ejemplo de prioridades dinámicas con:

$$Prio = \tau_{n+1}^{P}$$

- Desventaja: hambruna
- Variante: Aging (añejamiento)
- Se evita la hambruna aumentando cada cierto tiempo la prioridad de todos los procesos READY (**¡bonus!**)
- Cuando un proceso recibe la CPU recupera su prioridad original
- Con suficiente tiempo de permanencia en la cola, tarde o temprano un proceso ganará la prioridad suficiente para recibir la CPU

Round Robin

- Se inventó para los sistemas de tiempo compartido: utilizados por múltiples usuarios ***interactivos*** simultáneos
- Preemptive por definición
- Los procesos se turnan para recibir tajadas de tiempo de CPU de 10 a 100 milisegundos (*slices*)
- También se llama *time-slicing*
- *Minimiza el tiempo de respuesta*
- Informalmente se entiende tiempo de respuesta como el tiempo que transcurre desde que el usuario realiza una interacción con la aplicación hasta que recibe la primera señal de avance de su respuesta
- No considera la respuesta completa
- Debería ser el máximo tiempo transcurrido entre la entrega de cada resultado parcial

Round Robin: principio de funcionamiento

- Si un sistema tiene 5 usuarios activos, es mejor que cada usuario reciba 1/5 de la CPU de manera predecible que reciba CPU aleatoriamente
- Problema: podría no haber memoria suficiente para las 5 aplicaciones cargadas simultáneamente
- Alternativa: computadores personales más baratos, más lentos que los computadores institucionales pero con tiempo de respuesta predecible (años 80)
- Experimento: la salida de un comando consta de 10 líneas de información
- Un usuario no recibe ninguna respuesta hasta que a los 5 segundos aparecen las 10 líneas de una sola vez
- Otro usuario recibe una línea cada 1 segundo hasta completar el resultado completo a los 10 segundos
- ¿En qué situación se sentiría más conforme?

- Nuestra sicología prefiere la segunda

Round Robin: implementación

- Los procesos READY se mantienen en una cola FIFO (ready queue)
- Cuando el proceso en ejecución desocupa la CPU (pasa a estado WAIT), se extrae el proceso en primer puesto en la cola y se le cede la CPU
- ¿Qué hacer con los procesos que pasan a estado READY?
  - Se agregan al final de la cola: desfavorece a procesos intensivos en E/S
  - Se agregan al principio de la cola: hambruna para intensivos en CPU
  - Se otorga la CPU y el proceso que estaba ejecutándose se agrega al principio de la cola: hambruna
  - Idéntico, pero su tajada se reduce al tiempo que le restaba en el momento de pasar a estado de espera: sin hambruna pero con mayor cantidad de cambios de contexto implícitos

Tamaño de la tajada

- Mientras más grande, menor es el tiempo de respuesta
- Mientras más pequeña, mejor tiempo de respuesta pero mayor costo en cambios de contexto implícitos y mayor tiempo de despacho promedio
- Regla empírica: fijar el tamaño de tajada de modo que el 80% de las ráfagas duren menos que la tajada y por lo tanto se ejecutan sin cambios de contexto implícito