Planowanie przydziału procesora

1 Podstawowe pojęcia

Planowanie przydziału procesora [CPU scheduling] jest techniką pozwalającą zwiększyć wykorzystanie procesora, korzystając z wysokiego kosztu I/O w porównaniu do kosztu wykonania instrukcji procesora. Rozważmy proces, który korzysta z I/O. W prostym systemie, procesor czeka bezczynnie na zakończenie na zakończenie żądania. W systemie wieloprogramowym, czas ten jest spożytkowany na wykonywanie instrukcji innego programu.
Wykonanie procesu dzieli się na naprzemienne fazy procesora [CPU bursts] i fazy wejścia-wyjścia [I/O bursts].
Gdy procesor jest bezczynny, planista krótkoterminowy [short-term scheduler / CPU scheduler] wybiera proces z pamięci będący gotów do wykonywania i przydziela mu CPU.
Jeżeli planista podejmuje decyzje tylko gdy proces się kończy lub przełącza się do stanu czekania, mamy do czynienia z niewywyłaszczeniowym planowaniem przydziału [nonpreemptive scheduling / cooperating scheduling]. W przeciwnym wypadku (również gdy program przechodzi ze stanu wykonywania w stan gotowości bądź ze stanu oczekiwania do stanu gotowości) jest to planowanie wywłaszczeniowe [preemptive]. Planowanie wywłaszczeniowe wymaga dodatkowego wsparcia sprzętowego (np. timer) i może powodować szkodliwą rywalizację.
Ekspedytor [dispatcher] to moduł dający kontrolę nad CPU do procesu wybranego przez planistę krótkoterminowego. Zajmuje się on przełączaniem kontekstu, przełączaniem do trybu użytkownika i skokiem do odpowiedniej instrukcji programu. [Dispatch latency] to narzut spowodowany ekspedytora.

2 Kryteria planowania

- Użycie CPU - pożądane jest jak najwyższe, w praktyce od 40 do 90
- Przepustowość [throughput] - liczba procesów zakończonych w jednostce czasu
- Czas obiegu zadania [turnaround time] - czas od rozpoczęcia do zakończenia wybranego procesu
- Czas oczekiwania [waiting time] - czas spędzony w kolejce procesów gotowych do wykonania
- Czas odpowiedzi [response time] - czas od rozpoczęcia procesu do jego pierwszej odpowiedzi]

Pożądane jest zwiększenie użycia CPU i przepustowości, a zmniejszenie czasu obiegu zadania, czasu oczekiwania i czasu odpowiedzi. W zależności od potrzeb są różne priorytety.

3 Algorytmy planowania

- FCFS [first come - first served] (niewywłaszczeniowy) Jak sama nazwa wskazuje, procesy są wykonywane kolejno. Niezbyt mądry, patrz: diagram Gantta (lepiej by było najpierw wykonać P2 i P3). Efekt konwoju - kiedy krótkie procesy czekają na zakończenie dłuższego, tak jak niżej.

- SJF [shortest job first] Przydziela każdemu procesowi przewidywany czas kolejnej fazy procesora (np. średnia wykładnicza poprzednich). Może być niewywłaszczeniowy lub wywłaszczeniowy (wtedy [shortest remaining time first]. Jest optymalny pod względem średniej, gdybyśmy potrafili dokładnie podać długość procesów.

- [priority scheduling] Wykonujemy najpierw procesy o ważniejszym priorytecie (w zależności od konwencji jest to liczba mniejsza/większa). Np. SJF to [priority scheduling] dla priorytetu będącego odwrotnym do długości fazy procesora. Priorytety mogą być dobierane w różny sposób, zewnętrznie (np. typ procesu) lun wewnętrznie (np. liczba otwartych plików). Istnieją zarówno nie- jak i wywłaszczeniowe wersje tego algorytmu. Problemem takiego planity jest występowanie nieskończonego blokowania [indefinite blocking] (proces jest wypychany przez ważniejsze, choć nowsze procesy) i głodzenie [starvation]. Rozwiązaniem jest postarzanie [aging], tj. wzrost ważności procesu wraz z jego wiekiem.

- Planowanie rotacyjne [round-robin scheduling] Zaprojektowany dla systemów z podziałem czasu [time sharing systems]. Podobny do FCFS, ale dodano wywłaszczenie by zmieniać procesy po małej jednostce czasu (kwant czasu [time slice]). Planista idzie cyklicznie po kolejce gotowych procesów. Kwant czasu nie może być zbyt mały, by przełączanie kontekstu nie było zbyt drogie.

- Kolejki wielopoziomowe [Multilevel queue scheduling] Zaprojektowany dla procesów, które można zakwalifikować do różnych kategorii. Np. zadania pierwszo- i drugoplanowe. Dzielimy kolejkę gotowych procesów na kolejki dla poszczególnych kategorii, każda z nich może implementować dowolnie wybrany z algorytmów. Na przykład kolejka procesów systemowych może implementować planowanie rotacyjne, a procesów użytkownika - SJF. Kolejki mogą być wykonywane jedna po drugiej (niższe są wykonywane dopiero gdy wyższe są wolne) lub na zasadzie podziału czasu.

- Kolejki wielopoziomowe ze sprzężeniem zwrotnym [Multilevel feedback queue scheduling] Jak wyżej, tyle że można przesuwać procesy - np. kolejki RR z rosnącymi długościami kwantów, gdzie spychamy niżej niezakończone procesy. Takie kolejki zadane są przez szereg parametrów: liczba kolejek, algorytm planowania każdej kolejki, zasady awansowania/degradowania procesów, zasady określania, w której kolejce znajduje się proces wymagający obsługi.

4 Planowanie wątków

Jak to opisano wcześniej, wątki mogą być [user-level] bądź [kernel-level].
W wypadku użycia [lightweight processes], mamy do czynienia z lokalnym planowaniem procesowym [process local scheduling] [process-contention scope], tj. biblioteka decyduje, który wątek będzie wykonywany na LWP.
W przypadku wątków [kernel-level], używane jest globalne planowanie systemowe [global system scheduling] [system-contention scope], tj. wszystkie wątki rywalizują ze sobą w puli systemu.

5 Planowanie dla wielu procesorów

Procesory jednorodne [homogenoues] - procesory są tego samego rodzaju (fizycznie). W wieloprzetwarzaniu asymetrycznym [asymmetric multiprocessing] jeden procesor odpowiada za planowanie, a pozostałe wykonują kod. Taki model jest prosty w użyciu.
Drugie podejście to wieloprzetwarzanie symetryczne [symmetric multiprocessing], gdzie każdy procesor planuje swoje zadania. Procesy mogą być w wspólnej kolejce, lub każdy procesor może mieć własną.
Zauważmy, że z racji użycia cache'u jest lepiej, gdy proces jest wykonywany na jednym i tym samym procesorze. Stąd koncept [processor affinity], to jest unikanie przenoszenia procesu na inny CPU. Jeśli mimo to system nie gwarantuje, że proces będzie się wykonywał na jednym procesorze, mówimy o [soft affinity]. W przeciwnym wypadku jest to [hard affinity].
Równoważenie obciążeń [Load balancing] to zagadnienie równego podziału obciążenia między jednostkami. Wiążą się z tym pojęcia [push migration], czyli mechanizmu, w którym pewien proces nadzoruje co jakiś czas równy przydział pracy i ewentualnie go poprawia oraz [pull migration], gdzie bezczynny procesor zabiera zadanie od bardziej obciążonego.
Obecnie popularna jest architektura, w której wiele rdzeni umieszczono na tym samym chipie, tzw. procesory wielordzeniowe. Są one oszczędniejsze i szybsze niż osobne procesory.
Dzięki użyciu [multithreaded processor cores] przeciwdziała się [memory stall], tj. biernemu oczekiwaniu na dane z pamięci. Każdemu rdzeniowi przypisuje się kilka wątków. Kiedy jeden czeka na dane, drugi wykonuje instrukcje, zwiększając użycie procesora. Są dwa podejścia do tej techniki. W modelu [coarse-grained], wątek jest wykonywany aż do wydarzenia wiążącego się z opóźnieniem, np. [memory stall]. Model [fine-graned] przewiduje cykliczną zmianę wątków.

6 Systemy czasu rzeczywistego

Systemy czasu rzeczywistego dzielimy na łagodne [soft real-time systems], gdzie nie ma gwarancji co do tego, kiedy proces zostanie wykonany, oraz systemy rygorystyczne [hard real-time systems], gdzie zadanie musi zostać wykonane przed deadlinem.
[Event latency] nazywamy ilość czasu, jaka mija od pojawienia się zdarzenia do czasu jego obsłużenia. I tak mamy [interrupt latency], czyli czas od zaistnienia zdarzenia do uruchomiania zadania, które je obsługuje oraz [dispatch latency], składające się z fazy konfliktowej [conflict phase] (wywłaszczenia obecnie działającego procesu bądź oczekiwania na zwolnienie zasobów wymaganych przez proces) i czasu zajętego przez ekspedytor.
Wszystkie systemy czasu rzeczywistego muszą implementować jakiś rodzaj [priority-based scheduling] by odpowiednio obsługiwać zdarzenia. Procesy w systemach czasu rzeczywistego są [periodic], to jest wymagają od procesora ustalonego okresu czasu p, podczas którego są obsługiwane. [rate] takiego zadania to liczba $1/p$. Dzięki temu, korzystając z techniki zwanej [admission control] planista odrzuca proces, wiedząc że go nie wykona na czas, lub gwarantuje jego wykonanie.
Algorytmy planowania:

- [Rate-monotonic scheduling] Jeśli mniej ważny proces jest wykonywany, a ważniejszy jest gotowy, następuje wywłaszczenie i wykonanie ważniejszego procesu. Im krótszy [period], tym wyższy priorytet.
- [Earliest-deadline-first scheduling]
- [Proportional shares scheduling] Procesor przydziela procesom [shares], a następnie udziela im czasu proporcjonalnie do ich liczby.

7 Ocena algorytmów

Modelowanie deterministyczne - przyjmuje się z góry pewne obciążenie i określa dla niego działanie każdego algorytmu.
Inne metody: modele obsługi kolejek i implementacja.
Wzór Little'a: $n = \lambda * W$, gdzie W - średni czas oczekiwania procesu w kolejce, λ - tempo przybywania nowych procesów do kolejki, n - średnia długość kolejki.