Zarządzanie procesami w Linux 2.4.7 - zadania

Maciej Makowski

3 lutego 2013

1 Zadania

Zadania oznaczone (*) zostały wykorzystane na ćwiczeniach w dniu 23.11.2001

1. *(*)* Czy istnieje górne ograniczenie na priorytet zwykłego procesu (czyli wartość pola `counter` w `task_struct`)? Jeśli tak, to ile wynosi?

2. Funkcja `__wake_up_common(wait_queue_head_t *q, unsigned int mode, int nr_exclusive, const int sync)` ma za zadanie:

   - dla `nr_exclusive == 0` obudzić wszystkie procesy śpiące w kolejce wskazywanej przez `q`,
   - dla `nr_exclusive > 0` obudzić wszystkie te procesy śpiące w kolejce wskazywanej przez `q`, które nie mają ustawionej flagi `WQ_FLAG_EXCLUSIVE` oraz pierwsze `nr_exclusive` procesów z ustawioną tą flagą.

   Implementacja w Linuksie 2.4.7 zachowuje się zgodnie z tą semantyką pod warunkiem, że procesy są wstawiane do `q` w ten sposób, żeby wszystkie procesy z ustawioną flagą `WQ_FLAG_EXCLUSIVE` znalazły się na końcu kolejki (ten postulat realizują funkcje `add_wait_queue()`, `add_wait_queue_exclusive()` oraz `sleep_on()`). Jak poprawić `__wake_up_common()`, aby dawała wynik zgodny z zamierzonym nawet, jeśli procesy „ekskluzywne" i „nieekskluzywne" są w kolejce przemieszane?

3. *(*)* W systemie jednoprocesorowym działają trzy procesy obliczeniowe (nie wykonujące operacji wejścia/wyjścia): p, n_1, n_2. Procesy n wykonują nieskończoną pętlę, podczas gdy p wykonuje obliczenie wymagające $10 * q$ czasu procesora (q jest domyślną długością kwantu przyznawanego zwykłym procesom). Jak zmieniłby się łączny czas wykonania zadania liczonego przez p, gdyby podzielić je między trzy procesy p_1, p_2, p_3? Dla ustalenia uwagi załóżmy, że w początkowym stanie systemu procesy p znajdują się na początku kolejki procesów gotowych, zaś wartości `counter` i `NICE_TO_TICKS(nice)` wszystkich procesów wynoszą q.

4. *(*)* W systemie jednoprocesorowym działa k procesów czasu rzeczywistego $p_1 ... p_k$ (o takich samych wartościach `rt_priority`) szeregowanych metodą `SCHED_FIFO`, oraz pewna liczba zwykłych procesów. Każdy z procesów czasu rzeczywistego cyklicznie:

   - wykonuje się na procesorze przez czas t_{cpu}
   - oczekuje na wejście/wyjście przez czas t_{io}

   (a) jaka relacja powinna zachodzić między k, t_{cpu} i t_{io}, aby zwykłe procesy mogły w ogóle korzystać z procesora?

   (b) w jaki sposób zmieniłaby się sytuacja zwykłych procesów (miałyby dla siebie więcej czy mniej czasu procesora), gdyby procesy czasu rzeczywistego szeregować metodą `SCHED_RR`?

2 Rozwiązania

1. Wartość zmiennej `counter` wzrasta jedynie podczas przeliczania na nowo priorytetów procesów. Wówczas jest ona aktualizowana zgodnie ze wzorem

$$counter := \frac{counter}{2} + priority$$

gdzie $priority$ jest odpowiednio przeskalowaną wartością zmiennej `nice`. Przy założeniu, że proces w ogóle nie zużywa przyznanego mu czasu, wartość zmiennej `counter` będzie przy kolejnych przeliczaniach wartością kolejnych wyrazów szeregu geometrycznego:

$$priority,\ priority + \frac{priority}{2},\ priority + \frac{priority}{2} + \frac{priority}{4}...$$

zaś

$$\sum_{i=0}^{\infty} \frac{priority}{2^i} = 2 * priority$$

(ze wzoru na sumę szeregu geometrycznego), stąd górne ograniczenie na wartość zmiennej `counter` wynosi $2 * priority$.

2. *Rozwiązanie zostało zakodowane w C, ze względu na zwartą formę, jaką oferują konstrukcje tego języka. Możliwe jest oczywiście przedstawienie go w postaci pseudokodu - podobnie jak w trakcie prezentacji przedstawiony został oryginalny algorytm.*
Jeśli nie założymy odpowiedniego rozmieszczenia procesów, to w celu uzyskania zamierzonego efektu trzeba będzie przeglądać kolejkę do samego końca. Możliwa jest np. taka poprawka: wprowadzamy nową zmienną `int fin = 0;`, a następnie w miejscu najgłębszego zagnieżdżenia zamiast

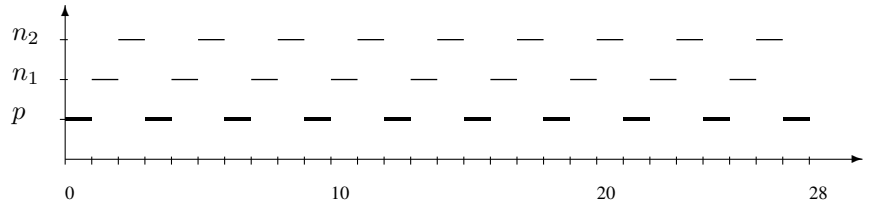
```
if (try_to_wake_up(p, sync) && (curr->flags&WQ_FLAG_EXCLUSIVE)
        && !--nr_exclusive)
    break;
```

wstawiamy

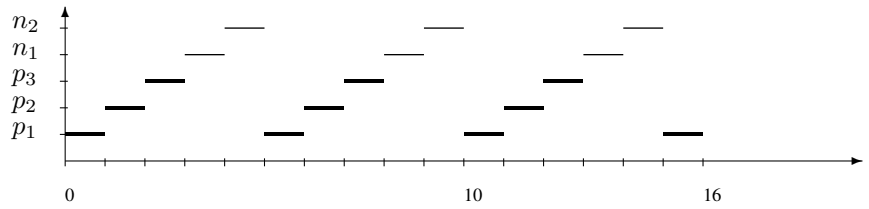
```
if (!fin && try_to_wake_up(p, sync) &&
        (curr->flags&WQ_FLAG_EXCLUSIVE) && !--nr_exclusive)
    fin = 1;
if (fin && !(curr->flags&WQ_FLAG_EXCLUSIVE))
    try_to_wake_up(p, sync);
```

Przy poprawianiu należy zwrócić uwagę, aby nie zmniejszyć `nr_exclusive` zanim nie będziemy mieli pewności, że `try_to_wake_up()` dla „ekskluzywnego" procesu się powiodło.

3. Wykres przydziału procesora dla pierwszego przypadku (obliczenie wykonywane przez jeden proces):

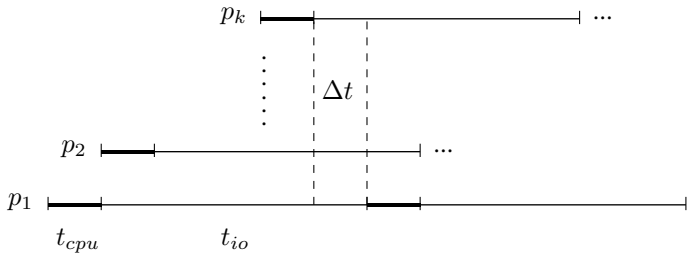


jak widać, wykonywanie procesu p zakończy się po czasie $28q$.
Dla drugiego przypadku (obliczenie wykonywane przez trzy procesy), wykres przydziału procesora wygląda następująco:



tutaj obliczenie zakończy się po czasie $16q$.
Wniosek: nawet w systemie jednoprocesorowym podział obliczenia między kilka procesów może skrócić czas wykonania - kosztem innych procesów działających w systemie.

4. (a) Schemat działania procesów czasu rzeczywistego:



Δt oznacza czas jaki upływa od momentu uśpienia p_k do obudzenia p_1. Ten właśnie odcinek czasu mają do dyspozycji zwykłe procesy, konieczne jest zatem, aby $\Delta t > 0$. Ponieważ $\Delta t = t_{io} - (k-1) * t_{cpu}$, stąd szukana zależność jest postaci

$$t_{io} - (k-1) * t_{cpu} > 0$$

(b) wystarczy zauważyć, że strategia `SCHED_FIFO` jest z punktu widzenia zwykłych procesów najgorszą możliwą, ponieważ odcinek czasu, w którym wszystkie procesy czasu rzeczywistego oczekują na wejście/wyjście jest najkrótszy. Należy pokazać, że przez czas t_{io} uśpienia jednego z procesów p, pozostałe procesy czasu rzeczywistego nie zużyją więcej niż $(k-1) * t_{cpu}$ czasu procesora. Przyjmijmy przeciwnie: w tej sytuacji któryś z $k-1$ procesów musiał w odcinku czasu długości t_{io} wykonywać się przez czas dłuższy niż t_{cpu}, co jest sprzeczne z definicją procesu czasu rzeczywistego zawartą w treści zadania. Ponieważ przy szeregowaniu `SCHED_FIFO` pozostałe $k-1$ procesów zużywa dokładnie $(k-1) * t_{cpu}$ czasu procesora, stąd ta strategia szeregowania jest, z punktu widzenia zwykłych procesów, najgorsza.

Pozostaje pokazać, że SCHED_RR może osiągnąć lepszy wynik; w tym celu wystarczy rozpatrzyć przykład, w którym długość kwantu dla SCHED_RR wynosi $t_{cpu}/2$. W takim przypadku czas Δt, w którym wszystkie procesy czasu rzeczywistego czekają, wyniesie $t_{io} - (k-1) * t_{cpu}/2$, co jest ostro większe od $t_{io} - (k-1) * t_{cpu}$.
Ogólnie, efektywność (z punktu widzenia zwykłych procesów) szeregowania SCHED_RR zależy od długości kwantu przydzielanego procesom czasu rzeczywistego, jednak nigdy nie jest gorsza niż w przypadku SCHED_FIFO.