BSD - alternatywa dla Linuksa

Różnice i podobieństwa w implementacji jądra (na przykładzie schedulera)

Maria Donten
Bartłomiej Etenkowski
Aleksander Zabłocki

Systemy Operacyjne 2006

Plan

Scheduler w Linuksie

O tym była mowa na wykładzie, na który wszyscyśmy pilnie chodzili.

Plan

Informacje ogólne

- w starszych wersjach kod schedulera był rozrzucony po całym jądrze
- uporządkowano to dopiero w wersji 5.0 (zachowano jednak ten sam algorytm)
- domyślny do wersji 5.1

Informacje ogólne

- w starszych wersjach kod schedulera był rozrzucony po całym jądrze
- uporządkowano to dopiero w wersji 5.0 (zachowano jednak ten sam algorytm)
- domyślny do wersji 5.1

Informacje ogólne

- w starszych wersjach kod schedulera był rozrzucony po całym jądrze
- uporządkowano to dopiero w wersji 5.0 (zachowano jednak ten sam algorytm)
- domyślny do wersji 5.1

Zasada działania

- 0 - 127 – procesy jądra
  128 - 159 – procesy czasu rzeczywistego
  160 - 223 – procesy użytkownika
  224 - 225 – procesy jałowe (idle)
- 64 dwukierunkowe listy procesów
- wątek w stanie runnable jest umieszczany w jednej z kolejek (wybór w zależności od priorytetu)
- wybór – pierwszy wątek z kolejki procesów z najwyższym priorytetem
- wątek wywłaszczony wraca na koniec swojej kolejki
- proces jest wyrzucany z kolejki, jeśli zaśnie np. na semaforze

Zasada działania

- 0 - 127 – procesy jądra
  128 - 159 – procesy czasu rzeczywistego
  160 - 223 – procesy użytkownika
  224 - 225 – procesy jałowe (idle)
- 64 dwukierunkowe listy procesów
- wątek w stanie runnable jest umieszczany w jednej z kolejek (wybór w zależności od priorytetu)
- wybór – pierwszy wątek z kolejki procesów z najwyższym priorytetem
- wątek wywłaszczony wraca na koniec swojej kolejki
- proces jest wyrzucany z kolejki, jeśli zaśnie np. na semaforze

Zasada działania

- 0 - 127 – procesy jądra
  128 - 159 – procesy czasu rzeczywistego
  160 - 223 – procesy użytkownika
  224 - 225 – procesy jałowe (idle)
- 64 dwukierunkowe listy procesów
- wątek w stanie runnable jest umieszczany w jednej z kolejek (wybór w zależności od priorytetu)
- wybór – pierwszy wątek z kolejki procesów z najwyższym priorytetem
- wątek wywłaszczony wraca na koniec swojej kolejki
- proces jest wyrzucany z kolejki, jeśli zaśnie np. na semaforze

Zasada działania

- 0 - 127 – procesy jądra
  128 - 159 – procesy czasu rzeczywistego
  160 - 223 – procesy użytkownika
  224 - 225 – procesy jałowe (idle)
- 64 dwukierunkowe listy procesów
- wątek w stanie runnable jest umieszczany w jednej z kolejek (wybór w zależności od priorytetu)
- wybór – pierwszy wątek z kolejki procesów z najwyższym priorytetem
- wątek wywłaszczony wraca na koniec swojej kolejki
- proces jest wyrzucany z kolejki, jeśli zaśnie np. na semaforze

Zasada działania

- 0 - 127 – procesy jądra
  128 - 159 – procesy czasu rzeczywistego
  160 - 223 – procesy użytkownika
  224 - 225 – procesy jałowe (idle)
- 64 dwukierunkowe listy procesów
- wątek w stanie runnable jest umieszczany w jednej z kolejek (wybór w zależności od priorytetu)
- wybór – pierwszy wątek z kolejki procesów z najwyższym priorytetem
- wątek wywłaszczony wraca na koniec swojej kolejki
- proces jest wyrzucany z kolejki, jeśli zaśnie np. na semaforze

Zasada działania

- 0 - 127 – procesy jądra
  128 - 159 – procesy czasu rzeczywistego
  160 - 223 – procesy użytkownika
  224 - 225 – procesy jałowe (idle)
- 64 dwukierunkowe listy procesów
- wątek w stanie runnable jest umieszczany w jednej z kolejek (wybór w zależności od priorytetu)
- wybór – pierwszy wątek z kolejki procesów z najwyższym priorytetem
- wątek wywłaszczony wraca na koniec swojej kolejki
- proces jest wyrzucany z kolejki, jeśli zaśnie np. na semaforze

Zasada działania

- na sztywno ustawiony kwant czasu – 0,1 s
- co cztery tyknięcia zegara (jakieś 0,4 s) przeliczane są priorytety dla procesów „żywych”
- co sekundę procesy są na nowo kolejkowane (koszt czasowy O(n)!)
- co tyknięcie jest przeliczana wartość `kg_estcpu` dla procesów żywych...
- a dla uśpionych po przebudzeniu

Zasada działania

- na sztywno ustawiony kwant czasu – 0,1 s
- co cztery tyknięcia zegara (jakieś 0,4 s) przeliczane są priorytety dla procesów „żywych”
- co sekundę procesy są na nowo kolejkowane (koszt czasowy O(n)!)
- co tyknięcie jest przeliczana wartość `kg_estcpu` dla procesów żywych...
- a dla uśpionych po przebudzeniu

Zasada działania

- na sztywno ustawiony kwant czasu – 0,1 s
- co cztery tyknięcia zegara (jakieś 0,4 s) przeliczane są priorytety dla procesów „żywych”
- co sekundę procesy są na nowo kolejkowane (koszt czasowy O(n)!)
- co tyknięcie jest przeliczana wartość `kg_estcpu` dla procesów żywych...
- a dla uśpionych po przebudzeniu

Zasada działania

- na sztywno ustawiony kwant czasu – 0,1 s
- co cztery tyknięcia zegara (jakieś 0,4 s) przeliczane są priorytety dla procesów „żywych”
- co sekundę procesy są na nowo kolejkowane (koszt czasowy O(n)!)
- co tyknięcie jest przeliczana wartość `kg_estcpu` dla procesów żywych...
- a dla uśpionych po przebudzeniu

Zasada działania

- na sztywno ustawiony kwant czasu – 0,1 s
- co cztery tyknięcia zegara (jakieś 0,4 s) przeliczane są priorytety dla procesów „żywych”
- co sekundę procesy są na nowo kolejkowane (koszt czasowy O(n)!)
- co tyknięcie jest przeliczana wartość `kg_estcpu` dla procesów żywych...
- a dla uśpionych po przebudzeniu

Plan

Informacje ogólne

- W schedulerze `4BSD` co sekundę trzeba było przechodzić listę wszystkich procesów – liniowy koszt czasowy. Scheduler `ULE` swoją pracę wykonuje w czasie stałym.
- wprowadzony w wersji 5.0, domyślny od 5.2
- mimo lepszego kosztu czasowego często mniej wydajny niż `4BSD`

Informacje ogólne

- W schedulerze `4BSD` co sekundę trzeba było przechodzić listę wszystkich procesów – liniowy koszt czasowy. Scheduler `ULE` swoją pracę wykonuje w czasie stałym.
- wprowadzony w wersji 5.0, domyślny od 5.2
- mimo lepszego kosztu czasowego często mniej wydajny niż `4BSD`

Informacje ogólne

- W schedulerze `4BSD` co sekundę trzeba było przechodzić listę wszystkich procesów – liniowy koszt czasowy. Scheduler `ULE` swoją pracę wykonuje w czasie stałym.
- wprowadzony w wersji 5.0, domyślny od 5.2
- mimo lepszego kosztu czasowego często mniej wydajny niż `4BSD`

Zasada działania

- dla każdego procesora trzy kolejki priorytetowe (`struct runq`): `run`, `next` i `idle`.
- wykonywane są procesy z kolejki `run` (dopóki nie opustoszeje, wtedy zamieniana jest z kolejką `next`)
- procesy z kolejki `idle` wykonywane są tylko wtedy, kiedy kolejki `run` oraz `next` są puste
- procesy o priorytecie < 160 oraz procesy interaktywne po wykonaniu wracają do kolejki `run`, pozostałe lądują w kolejce `next`
- współczynnik interaktywności jest niezależny od `nice`
- próg interaktywności jest ustawiony na sztywno w `sched_ule.c` (i jest równy 30).

Zasada działania

- dla każdego procesora trzy kolejki priorytetowe (`struct runq`): `run`, `next` i `idle`.
- wykonywane są procesy z kolejki `run` (dopóki nie opustoszeje, wtedy zamieniana jest z kolejką `next`)
- procesy z kolejki `idle` wykonywane są tylko wtedy, kiedy kolejki `run` oraz `next` są puste
- procesy o priorytecie < 160 oraz procesy interaktywne po wykonaniu wracają do kolejki `run`, pozostałe lądują w kolejce `next`
- współczynnik interaktywności jest niezależny od `nice`
- próg interaktywności jest ustawiony na sztywno w `sched_ule.c` (i jest równy 30).

Zasada działania

- dla każdego procesora trzy kolejki priorytetowe (`struct runq`): `run`, `next` i `idle`.
- wykonywane są procesy z kolejki `run` (dopóki nie opustoszeje, wtedy zamieniana jest z kolejką `next`)
- procesy z kolejki `idle` wykonywane są tylko wtedy, kiedy kolejki `run` oraz `next` są puste
- procesy o priorytecie < 160 oraz procesy interaktywne po wykonaniu wracają do kolejki `run`, pozostałe lądują w kolejce `next`
- współczynnik interaktywności jest niezależny od `nice`
- próg interaktywności jest ustawiony na sztywno w `sched_ule.c` (i jest równy 30).

Zasada działania

- dla każdego procesora trzy kolejki priorytetowe (`struct runq`): `run`, `next` i `idle`.
- wykonywane są procesy z kolejki `run` (dopóki nie opustoszeje, wtedy zamieniana jest z kolejką `next`)
- procesy z kolejki `idle` wykonywane są tylko wtedy, kiedy kolejki `run` oraz `next` są puste
- procesy o priorytecie < 160 oraz procesy interaktywne po wykonaniu wracają do kolejki `run`, pozostałe lądują w kolejce `next`
- współczynnik interaktywności jest niezależny od `nice`
- próg interaktywności jest ustawiony na sztywno w `sched_ule.c` (i jest równy 30).

Zasada działania

- dla każdego procesora trzy kolejki priorytetowe (`struct runq`): `run`, `next` i `idle`.
- wykonywane są procesy z kolejki `run` (dopóki nie opustoszeje, wtedy zamieniana jest z kolejką `next`)
- procesy z kolejki `idle` wykonywane są tylko wtedy, kiedy kolejki `run` oraz `next` są puste
- procesy o priorytecie < 160 oraz procesy interaktywne po wykonaniu wracają do kolejki `run`, pozostałe lądują w kolejce `next`
- współczynnik interaktywności jest niezależny od `nice`
- próg interaktywności jest ustawiony na sztywno w `sched_ule.c` (i jest równy 30).

Zasada działania

- dla każdego procesora trzy kolejki priorytetowe (`struct runq`): `run`, `next` i `idle`.
- wykonywane są procesy z kolejki `run` (dopóki nie opustoszeje, wtedy zamieniana jest z kolejką `next`)
- procesy z kolejki `idle` wykonywane są tylko wtedy, kiedy kolejki `run` oraz `next` są puste
- procesy o priorytecie < 160 oraz procesy interaktywne po wykonaniu wracają do kolejki `run`, pozostałe lądują w kolejce `next`
- współczynnik interaktywności jest niezależny od `nice`
- próg interaktywności jest ustawiony na sztywno w `sched_ule.c` (i jest równy 30).

Sprawiedliwy podział zadań między procesorami

- *interprocessor interrupt* – procesor, który nie ma w danej chwili nic do roboty, przejmuje zadanie od innego procesora
- `sched_balance.c` – funkcja wykonywana dwa razy na sekundę – procesory z największą i najmniejszą liczbą procesów w kolejce `run` dzielą się zadaniami

Sprawiedliwy podział zadań między procesorami

- *interprocessor interrupt* – procesor, który nie ma w danej chwili nic do roboty, przejmuje zadanie od innego procesora
- `sched_balance.c` – funkcja wykonywana dwa razy na sekundę – procesory z największą i najmniejszą liczbą procesów w kolejce `run` dzielą się zadaniami