Systemy komputerowe

Lista zadań nr 12

Na ćwiczenia 1. i 2. czerwca 2022

Zadanie 1. W systemie twardego czasu rzeczywistego (ang. *hard real-time system*) chcemy wykonać cztery procesy, z okresami 50, 100, 200 i 250 ms. (liczba p) odpowiednio. Czasy obsługi tych procesów (liczba t) to 35, 20, 10 i x odpowiednio. Jaka jest maksymalna wartość x, dla której będzie to możliwe? Jak zmieni się ta odpowiedź, gdy będziemy mieli do czynienia z systemem miękkiego czasu rzeczywistego?

Zadanie 2. W systemie twardego czasu rzeczywistego chcemy wykonać dwa procesy, z okresami 50 i 75 ms. odpowiednio. Czasy obsługi tych procesów to 25 i 30 ms.

a) Czy jest to możliwe przy użyciu algorytmu RMS (ang. *rate-monotonic scheduling*)? Odpowiedź zilustruj diagramem Gantta.
b) Powtórz punkt a) używając algorytmu EDF (ang. *earliest deadline first*).

Zadanie 3 (2pkty). Oto opis algorytmu planowania zadań w jednym z zamierzchłych wydań systemu Linuks. Każdy z procesów p przy wprowadzaniu do systemu ma ustalone i niezmienne w czasie: **statyczny priorytet** (zapisany w p->priority) oraz **typ procesu** (zapisany w p->type. Np. SHED_REALTIME – proces miękkiego czasu rzeczywistego, SHED_NORMAL – pozostałe procesy). Ma również wartość **nice** (zapisaną w p->nice), dzięki której użytkownik może wpływać na priorytet procesu podczas wykonania. Czas procesora podzielony jest na **epoki**. Każdy proces p, który po raz pierwszy w bieżącej epoce przeszedł w stan READY lub znajdował się w tym stanie w momencie jej rozpoczęcia, otrzyma kwant czasu o zmiennej długości. Jego wartość reprezentowana jest w postaci liczby tyknięć zegara przez którą proces może działać, a zapisana jest w bloku kontrolnym procesu, w zmiennej p->counter. Ta wartość obliczana jest następująco p->counter = (p->counter >> 1) + NICE_TO_TICKS(p->nice). Gdy zajdzie potrzeba wybrania następnego procesu do wykonania, algorytm aktualizuje zmienną p->counter aktualnego procesu, odejmując od niej liczbę zużytych tyknięć, oraz przeszukuje kolejkę procesów gotowych i wybiera z niej proces o najwyższej wartości funkcji goodness() zdefiniowanej tak:

```
unsigned int goodness(p) {
 if (p->policy == SCHED_REALTIME)
  return 1000 + p->priority;
 if (p->counter == 0)
  return 0;
 return p->counter + p->priority

}
```

Odpowiedz na następujące pytania:

a) kiedy powinno nastąpić przejście do kolejnej epoki?
b) w jakim czasie działa ten algorytm planowania zadań, w systemie z n procesami?
c) czy w tym algorytmie występuje **głodzenie** procesów?
d) wymień **wszystkie** typy procesów **faworyzowane** przez ten algorytm.

Odpowiedzi uzasadnij.

Zadanie 4. Przypomnij zasadę działania algorytmu planowania zadań "O(1)".

Wskazówka: Podrozdział 5.6.3 Abraham Silberschatz, Peter B. Galvin, and Greg Gagne. 2012. Operating System Concepts (**8th. ed.**). Uwaga: numer edycji jest istotny.

Zadanie 5. Przeczytaj podrozdziały 5.4.1 — 5.4.5 artykułu Aas, J. (2005). Understanding the Linux 2.6.8.1 CPU Scheduler i wyjaśnij, w jaki sposób w algorytmie "O(1)" wyliczany jest dla procesu dynamiczny priorytet i kwant czasu.

Zadanie 6. Załóżmy, że w systemie komputerowym koszt przełączenia zadania wynosi 0, a kwanty czasu przydzielane procesom mogą być dowolnie małe. Mówimy, że algorytm planowania zadań realizuje **perfekcyjną wielozadaniowość**, jeśli w każdym przedziale czasu ε każdy spośród n procesów chcących się wykonywać, będzie się wykonywać przez czas $\frac{\varepsilon}{n}$. Przypomnij zasadę działania algorytmu CFS (ang. *Completely Fair Scheduler*) i uzasadnij, że przybliża on perfekcyjną wielozadaniowość.

Wskazówka: Podrozdział 5.7.1 Abraham Silberschatz, Peter B. Galvin, and Greg Gagne. 2012. Operating System Concepts (**10th. ed.**). Uwaga: numer edycji jest istotny.

Zadanie 7 (2pkty). Przeczytaj Roberson, Jeff. (2003). ULE: a modern scheduler for FreeBSD. USENIX BSDCon. Vol. 3. i przedstaw w skrócie zasadę działania algorytmu ULE.