Systemy operacyjne

Lista zadań nr 12

Na zajęcia 14 stycznia 2021

Należy przygotować się do zajęć czytając następujące materiały:

- Tanenbaum (wydanie czwarte): 2.3
- Arpaci-Dusseau: 28 (Locks[1]), 31 (Semaphores[2]), 32 (Common Concurrency Problems[3])

UWAGA! W trakcie prezentacji należy być gotowym do zdefiniowania pojęć oznaczonych **wytłuszczoną** czcionką.

Zadanie 1. Zapoznaj się z poniższym programem. Rozważamy wartości przechowywane w zmiennych: «`myid`», «`strtab`», «`vargp`», «`cnt`», «`argc`» i «`argv[0]`». Określ czy są one **współdzielone** i które z nich będą źródłem **wyścigów** (ang. *data race*).

```
1  __thread long myid;
2  static char **strtab;
3
4  void *thread(void *vargp) {
5   myid = *(long *)vargp;
6   static int cnt = 0;
7   printf("[%ld]: %s (cnt=%d)\n", myid, strtab[myid], ++cnt);
8   return NULL;
9  }
10
11 int main(int argc, char *argv[]) {
12   ...
13   strtab = argv;
14   while (argc > 0) {
15     myid = --argc;
16     pthread_create(&tid, NULL, thread, (void *)&myid);
17   }
18   ...
19 }
```

Zadanie 2. Sekcja krytyczna (ang. *critical section*) to fragment kodu w programie współbieżnym posiadający pewne właściwości. Podaj i uzasadnij założenia jakie musi spełniać rozwiązanie problemu sekcji krytycznej (§2.3.2). Czemu w programach przestrzeni użytkownika do jej implementacji nie możemy używać **wyłączania przerwań** (ang. *interrupt disable*)? Odwołując się do Prawa Amdahla powiedz czemu programistom powinno zależeć na tym, by sekcje krytyczne były możliwie jak najkrótsze – określa się to również mianem **drobno-ziarnistego blokowania** (ang. *fine-grained locking*).

Zadanie 3. Podaj w pseudokodzie semantykę **instrukcji atomowej** `compare-and-swap` i przy jej pomocy zaimplementuj **blokadę wirującą** (ang. *spin lock*) (§28.7). Niech typ «`spin_t`» będzie równoważny «`int`». Podaj ciało procedur «`void lock(spin_t *)`» i «`void unlock(spin_t *)`». Czemu blokada wirująca nie jest **sprawiedliwa** (ang. *fair*) (§28.8)? Wiemy, że w przestrzeni użytkownika wątek może zostać wywłaszczony, jeśli znajduje się w sekcji krytycznej chronionej dowolną blokadą. Jakie problemy to rodzi?

Zadanie 4. Podaj cztery warunki konieczne do zaistnienia zakleszczenia. Na podstawie §32.3 wyjaśnij w jaki sposób można **przeciwdziałać zakleszczeniom** (ang. *deadlock prevention*)? Które z proponowanych rozwiązań stosuje się w praktyce (np. jądrze Linux i FreeBSD)? Czemu pozostałe nie znajdują zastosowań?

[1] `http://pages.cs.wisc.edu/~remzi/OSTEP/threads-locks.pdf`
[2] `http://pages.cs.wisc.edu/~remzi/OSTEP/threads-sema.pdf`
[3] `http://pages.cs.wisc.edu/~remzi/OSTEP/threads-bugs.pdf`

Zadanie 5. Poniżej znajduje się propozycja[4] programowego rozwiązania problemu **wzajemnego wykluczania** (ang. *mutual exclusion*) dla dwóch procesów. Znajdź kontrprzykład, w którym to rozwiązanie zawodzi.

```
1  shared boolean blocked [2] = { false, false };
2  shared int turn = 0;
3
4  void P (int id) {
5    while (true) {
6      blocked[id] = true;
7      while (turn != id) {
8        while (blocked[1 - id])
9          continue;
10       turn = id;
11     }
12     /* put code to execute in critical section here */
13     blocked[id] = false;
14   }
15 }
16
17 void main() { parbegin (P(0), P(1)); }
```

Ciekawostka: Okazuje się, że nawet recenzenci renomowanego czasopisma „Communications of the ACM" dali się zwieść.

Zadanie 6. Wiemy, że **aktywne czekanie** (ang. *busy waiting*) nie jest właściwym sposobem oczekiwania na zwolnienie blokady. Czemu oddanie czasu procesora funkcją «`yield`» (§28.13) nie rozwiązuje wszystkich problemów, które mieliśmy z blokadami wirującymi? Zreferuj implementację **blokad usypiających** podaną w §28.14. Czemu jest ona niepoprawna bez użycia funkcji «`setpark`»? Czy rozwiązuje problem **głodzenia**?

Zadanie 7. Poniżej podano błędną implementację **semafora zliczającego** przy pomocy **semaforów binarnych**. Jaka jest główna różnica między semaforem binarnym, a **muteksem**? Dopuszczamy, żeby «`count`» był liczbą ujemną, w takim przypadku jej wartość bezwzględna oznacza liczbę uśpionych procesów. Znajdź kontrprzykład i zaprezentuj wszystkie warunki niezbędne do jego odtworzenia.

```
1  struct csem {
2    bsem mutex;
3    bsem delay;
4    int count;
5  };
6
7  void csem::csem(int v) {
8    mutex = 1;
9    delay = 0;
10   count = v;
11 }
```

```
13 void csem::P() {
14   P(mutex);
15   count--;
16   if (count < 0) {
17     V(mutex);
18     P(delay);
19   } else {
20     V(mutex);
21   }
22 }
```

```
23 void csem::V() {
24   P(mutex);
25   count++;
26   if (count <= 0)
27     V(delay);
28   V(mutex);
29 }
```

UWAGA! Poniżej znajduje się zadanie przygotowujące do projektu „System plików".

Zadanie 8. Wykorzystując dane zawarte w deskryptorach grup bloków oraz superbloku podaj pseudokod algorytmów wyznaczających: zajętość i-tego i-węzła, zajętość k-tego bloku, adres (fizyczny) bloku zawierającego dane k-tego (logicznego) bloku pliku opisanego przez i-ty i-węzeł. Drugi algorytm powinien zwracać zero, jeśli k-ty blok pliku nie istnieje. Najpierw należy wczytać i-ty i-węzeł, a następnie odnaleźć adres k-tego bloku na podstawie tablicy «`i-block`» i bloków pośrednich, jeśli to niezbędne.

[4] Harris Hyman, „Comments on a Problem in Concurrent Programming Control", January 1966.