# Zadanie 4

**Otwarto:** poniedziałek, 15 maja 2023, 00:00
**Wymagane do:** piątek, 2 czerwca 2023, 23:59

Celem zadania jest zaimplementowanie w MINIX-ie strategii szeregowania „max before deadline” oraz dodanie wywołania systemowego, które umożliwi procesom wybór tej strategii.

Każdy proces szeregowany tym algorytmem określa termin zakończenia, przed którym chce zostać wykonany - parametr `deadline`, przewidywany całkowity czas jego wykonywania - parametr `estimate` oraz zachowanie w przypadku przekroczenia czasu - flaga `kill`. Wartość `deadline` jest liczbą całkowitą nieujemną określającą datę w milisekundach od godz. 00.00 01.01.1970 lub `-1` oznaczającą rezygnację z tego sposobu szeregowania. Wartość `estimate` jest liczbą całkowitą dodatnią określającą szacowany czas w milisekundach potrzebny do wykonania procesu. Procesy do wykonania są szeregowane w taki sposób, aby zmaksymalizować liczbę procesów wykonanych przed ich terminem zakończenia.

## Szeregowanie

Wszystkie procesy szeregowane zgodnie z algorytmem „max before deadline” mają ten sam ustalony priorytet `DEADLINE_Q`, co oznacza, że znajdują się w tej samej kolejce procesów gotowych do wykonania. W obrębie tej kolejki wybieramy pierwszy z oczekujących procesów. Kolejka ta jest modyfikowana zgodnie z następującymi zasadami:

- Zakładamy, że każdy proces wykona się dokładnie w swoim przewidywanym całkowitym czasie wykonania `estimate`.
- Kiedy kolejka jest pusta i do szeregowania zgłasza się nowy proces, to trafia na jej początek.
- Kiedy w kolejce znajduje się już jakiś proces i do szeregowania zgłasza się nowy o przewidywanym całkowitym czasie wykonania `estimate`, to trafia on na pozycję `p`, taką że:
  - jeżeli przed jego dodaniem `k` procesów mogło wykonać się przed swoim terminem zakończenia, to po jego dodaniu `k + 1` procesów powinno móc się wykonać przed swoim terminem zakończenia;
  - jeżeli pozycji spełniających powyższy warunek jest więcej niż jedna, to trafia on na ostatnią z nich, dla której proces na pozycji `p - 1` ma przewidywany całkowity czas wykonania mniejszy lub równy `estimate`, a proces na pozycji `p + 1` ma przewidywany całkowity czas wykonania większy lub równy `estimate`;
  - jeżeli taka pozycja nie istnieje, to proces trafia na ostatnią pozycję, taką że nadal te same `k` procesów może się wykonać przed swoim terminem zakończenia oraz proces na pozycji `p - 1` ma przewidywany całkowity czas wykonania mniejszy lub równy `estimate`, a proces na pozycji `p + 1` ma przewidywany całkowity czas wykonania większy lub równy `estimate`.
  - Przy wkładaniu procesu do kolejki, jeżeli aktualny czas przekracza jego termin zakończenia `deadline`, to proces ten powraca do szeregowania domyślnego, do kolejki, w której znajdował się przed rozpoczęciem szeregowania algorytmem "max before deadline".

Jeżeli okaże się, że wybrany proces wykonywał się już dłużej niż przewidywany całkowity czas jego wykonywania `estimate`, to wybierany jest kolejny proces, a przyszłość rozważanego procesu zależy od wartości flagi `kill`:

- jeżeli flaga `kill` jest ustawiona na `true`, to proces zostaje zabity;
- jeżeli flaga `kill` jest ustawiona na `false`, to proces powraca do szeregowania domyślnego, trafiając na koniec kolejki `PENALTY_Q`.

Podczas działania procesy szeregowane zgodnie z nowym algorytmem nie zmieniają swojego priorytetu (kolejki) w odróżnieniu od zwykłych procesów szeregowanych domyślnie. Należy zadbać o to, aby zwykłym procesom nie był przydzielany priorytet `DEADLINE_Q`.

Procesy-dzieci procesów szeregowanych metodą „max before deadline” dziedziczą po rodzicach wszystkie parametry związane z tym szeregowaniem (`deadline`, `estimate`, `kill`, ostatnią kolejkę sprzed zmiany szeregowania, czas wykonywania od momentu zmiany szeregowania).

Jako czas wykonywania rozumiemy czas spędzony przez proces w przestrzeni użytkownika.

## Implementacja

Implementacja powinna zawierać definicje stałych `DEADLINE_Q = 8` oraz `PENALTY_Q = 14`.

Implementacja powinna zawierać nową funkcję systemową

```
int sched_deadline(int64_t deadline, int64_t estimate, bool kill);
```

Jeśli wartość parametru `deadline` jest nieujemna, to szeregowanie procesu zostanie zmienione na algorytm „max before deadline” z terminem zakończenia `deadline` i przewidywanym całkowitym czasem wykonania `estimate`. Wartość `-1` oznacza, że proces rezygnuje z szeregowania algorytmem „max before deadline” i wraca do szeregowania domyślnego, do kolejki, w której znajdował się przed rozpoczęciem szeregowania tym algorytmem.

Funkcja powinna przekazywać jako wynik `0`, jeśli metoda szeregowania została zmieniona pomyślnie, a `-1` w przeciwnym przypadku. Jeśli wartości parametrów nie są prawidłowe, czyli wartość `deadline` jest mniejsza niż `now + estimate` i rożna od `-1`, gdzie `now` to bieżąca data, albo `estimate` ma wartość niedodatnią, to `errno` przyjmuje wartość `EINVAL`. Jeśli proces, który chce zmienić metodę szeregowania na „max before deadline”, jest już szeregowany zgodnie z tym algorytmem, to `errno` przyjmuje wartość `EPERM`. Podobnie powinno się stać, gdy proces, który chce zrezygnować z szeregowania „max before deadline”, wcale nie jest nim szeregowany.

Dopuszczamy zmiany w katalogach:

- `/usr/src/minix/servers/sched`,
- `/usr/src/minix/servers/pm`,
- `/usr/src/minix/kernel`,
- `/usr/src/lib/libc/misc`,
- `/usr/src/minix/lib/libsys`.

oraz w plikach nagłówkowych:

- `/usr/src/minix/include/minix/com.h` który będzie kopiowany do `/usr/include/minix/com.h`,
- `/usr/src/minix/include/minix/callnr.h`, który będzie kopiowany do `/usr/include/minix/callnr.h`,
- `/usr/src/include/unistd.h`, który będzie kopiowany do `/usr/include/unistd.h`,
- `/usr/src/minix/include/minix/syslib.h`, który będzie kopiowany do `/usr/include/minix/syslib.h`,
- `/usr/src/minix/include/minix/ipc.h`, który będzie kopiowany do `/usr/include/minix/ipc.h`,
- `/usr/src/minix/include/minix/config.h`, który będzie kopiowany do `/usr/include/minix/config.h`.

## Wskazówki

Do zmieniania metody szeregowania można dodać nową funkcję systemową mikrojądra. Warto w tym przypadku wzorować się na przykład na funkcji `do_schedule()`. Można też próbować zmodyfikować tę funkcję.

Przypominamy, że za wstawianie do kolejki procesów gotowych odpowiedzialne jest mikrojądro (`/usr/src/minix/kernel/proc.c`). Natomiast o wartości priorytetu decyduje serwer `sched`, który powinien dbać o to, aby zwykłym procesom nie przydzielić priorytetu `DEADLINE_Q`.

Nie trzeba (i nie jest zalecane) pisanie nowego serwera szeregującego. Można zmodyfikować domyślny serwer `sched`.

Aby nowy algorytm szeregowania zaczął działać, należy wykonać `make; make install` w katalogu `/usr/src/minix/servers/sched` oraz w innych katalogach zawierających zmiany. Następnie trzeba zbudować nowy obraz jądra, czyli wykonać `make hdboot` w katalogu `/usr/src/releasetools` i ponownie uruchomić system. Gdyby obraz nie chciał się załadować lub wystąpił poważny błąd (*kernel panic*), należy przy starcie systemu wybrać opcję 6, która załaduje oryginalne jądro.

Aktualny czas można sprawdzić np. funkcją `int getuptime(clock_t*, clock_t*, time_t*)`.

Chcąc uzyskać aktualny czas, będąc w jądrze, można wzorować się na implementacji `int do_times(struct proc*, message*)`.

## Rozwiązanie

Poniżej przyjmujemy, że `ab123456` oznacza identyfikator osoby rozwiązującej zadanie. Należy przygotować łatkę (ang. *patch*) ze zmianami w katalogu `/usr`. Plik o nazwie `ab123456.patch` uzyskujemy za pomocą polecenia `diff -rupNEZbB`, tak jak w zadaniu 3.

Rozwiązanie w postaci łatki `ab123456.patch` należy umieścić w Moodle.

Uwaga: nie przyznajemy punktów za rozwiązanie, w którym łatka nie nakłada się poprawnie, które nie kompiluje się lub powoduje *kernel panic* podczas uruchamiania.

W celu skompilowania i uruchomienia systemu z nową wersją szeregowania wykonane będą następujące polecenia:

```
cd /
patch -t -p1 < ab123456.patch
cd /usr/src; make includes
cd /usr/src/minix/kernel; make && make install
cd /usr/src/minix/lib/libsys; make && make install
cd /usr/src/minix/servers/sched; make && make install
cd /usr/src/minix/servers/pm; make && make install
cd /usr/src/lib/libc; make && make install
cd /usr/src/releasetools; make hdboot
```

Następnie nowo zbudowany system zostanie przetestowany na serii testów automatycznych. Ostateczny wynik rozwiązania będzie zależał od: stabilności systemu, poprawności i efektywności działania na testach automatycznych oraz stylu kodowania.

## Pytania

Pytania dotyczące zadania można zadawać w przeznaczonym do tego wątku na Moodle.

Dodaj pracę

## Status przesłanego zadania

| | |
|---|---|
| **Status przesłanego zadania** | Nie przesłano jeszcze zadania |
| **Stan oceniania** | Nieocenione |
| **Pozostały czas** | Pozostało 17 dni 6 godzin |

Pobierz aplikację mobilną