SYSTEMY OPERACYJNE – laboratorium

Szeregowanie procesów

Przygotował: Tomasz Bocheński

1. Treść zadania:

SZEREGOWANIE PROCESÓW WEDŁUG ALGORYTMU ROUND ROBIN Z PRIORYTETYZACJĄ

W ramach zadania należy zaimplementować szeregowanie procesów według algorytmu Round-Robin z priorytetyzacją.
Zasada działania algorytmu:
- procesy znajdują się w 3 kolejkach,
- każdy proces domyślnie znajduje się w kolejce nr 1 (najwyższy priorytet),
- SO przydziela procesom zawsze jeden kwant czasu,
- w pierwszej kolejności kwant czasu otrzymują procesy z kolejki nr 1 (najwyższy priorytet),
- gdy wszystkie procesy z kolejki nr 1. otrzymały po jednym kwancie czasu, kwant czasu przydzielany jest pierwszemu procesowi w kolejce nr 2,
- po każdym przyznaniu kwantu czasu jednemu z procesów w kolejce nr 2 czas procesora przydzielany jest ponownie wszystkim procesom w kolejce nr 1 (za wyjątkiem poniższej sytuacji), a następnie kolejnemu w kolejce 2,
- gdy wszystkie procesy z kolejki nr 2 otrzymają po jednym kwancie czasu, kwant czasu zostaje przyznany pierwszemu procesowi w kolejce nr 3, itd.

Dodatkowo proszę o zaimplementowanie wywołań systemowych, które pozwolą na:
- sprawdzenie numeru kolejki, w której znajduje się, proces o zadanym numerze pid.
- przeniesienie procesu o zadanym numerze pid do kolejki o zadanym numerze,

2. Wady i zalety powyższego rozwiązania:

Zalety:

- prosty w zrozumieniu działania;
- możliwość podzielenia procesów na procesy o większym priorytecie (kolejka nr 1), średnim priorytecie (kolejka nr 2) oraz małym priorytecie (kolejka nr 3);

Wady:

- w ramach konkretnej kolejki procesy traktowane są tak samo, nie można w żaden sposób wyznaczyć w niej procesu który jest ważniejszy od reszty i powinien być wykonywany częściej;
- problem z doborem odpowiedniej długości trwania kwantu czasu:
  - za krótkie kwanty czasu powodują zbyt częste przełączanie kontekstu, zmniejszając wydajność procesora;
  - za długie kwanty czasy powodują długi czas reakcji;

3. Opis implementacji – w których plikach co zostanie zmienione:

Założenie: wszystkie procesy przechowywane będą w jednej kolejce. Zmienna systemowa NR_KOLEJKI oznaczać będzie numer 'abstrakcyjnej' kolejki w której znajduje się proces. Zaimplementowane dalej algorytmy służyć będą do poprawnej obsługi realnej kolejki składającej się z procesów znajdujących się w trzech 'abstrakcyjnych' kolejkach, tak jak to jest w treści zadania.

Plik usr/src/kernel.proc.h:
W pliku tym znajduje się tabela procesów, w której zawarte są różne informacje o procesie.

- w strukturze proc należy dodać dodatkową zmienną systemową **NR_KOLEJKI**, która przyjmowałaby wartości 1, 2 lub 3 reprezentujące numer abstrakcyjnej kolejki w której się proces;

Plik usr/scr/kernel/proc.c:
W pliku tym znajdują się definicje funkcji przeznaczonych do obsługi procesów.

- w funkcji sched() należy zaimplementować metodę szeregowania przedstawioną w treści zadania. Oto moja propozycja algorytmu modyfikującego funkcję sched:

Niech proces wykonany znajdujący się na początku kolejki będzie nazywał się HEAD

1) szukam pierwszego procesu z numerem kolejki większym od numeru kolejki procesu HEAD zaczynając od drugiej pozycji kolejki;
- jeśli taki proces nie istnieje, to proces HEAD wstawiany jest na koniec kolejki;

2) wyznaczam potencjalną pozycje w kolejce dla procesu HEAD poprzez przejście na kolejną pozycje (wtedy proces ten znajduje się za pierwszym procesem z innej kolejki);
- jeśli kolejna pozycja kolejki nie istnieje, proces HEAD wstawiany jest na koniec kolejki;

3) sprawdzam przypadek szczególny: numer kolejki procesu HEAD wynosi 1 i numer kolejki procesu znajdującego się na wyznaczonej wcześniej potencjalnej pozycji wynosi 3 oraz numer kolejki procesu znajdującego się jedną pozycję przed potencjalną pozycją wynosi 2. Jeżeli przypadek ten nie występuje mamy do czynienia z sytuacją opisaną w treści zadania „po każdym przyznaniu kwantu czasu jednemu z procesów w kolejce nr 2 czas procesora przydzielany jest ponownie wszystkim procesom w kolejce nr 1 (za wyjątkiem poniższej sytuacji), a następnie kolejnemu w kolejce 2”. Jeśli przypadek ten wystąpi mamy do czynienia z sytuacją „gdy wszystkie procesy z kolejki nr 2 otrzymają po jednym kwancie czasu, kwant czasu zostaje przyznany pierwszemu procesowi w kolejce nr 3” (opisaną w poprzednim cytacie słowami „ za wyjątkiem poniższej sytuacji”). W tym przypadku nową potencjalną pozycją staje się kolejna pozycja za „starą” pozycją potencjalną;
- jeśli kolejna pozycja kolejki nie istnieje, to proces HEAD wstawiany jest na koniec kolejki;

4) wyznaczam ostateczną pozycję dla procesu HEAD poprzez znalezienie pierwszej pozycji z numerem kolejki większym od numeru kolejki HEAD poczynając od potencjalnej pozycji;
- jeśli taka pozycja nie istnieje, to proces HEAD wstawiany jest na koniec kolejki;

PSEUDOKOD
Jest to tylko pseudokod reprezentujący algorytm szeregowania w kolejce. W rzeczywistości żeby wstawić proces w określone miejsce w kolejce będzie potrzebny wskaźnik na proces poprzedni.

```
tmp = HEAD->p_nextready;
/// punkt 1
while(tmp != NIL_PROC && HEAD->NR_KOLEJKI >= tmp->NR_KOLEJKI)
        tmp = tmp->p_nextready;
if (tmp == NIL_PROC)
        wstaw na koniec kolejki i zakończ / ustaw odpowiednia flage;
/// punkt 2
tmp2 = tmp;
tmp = tmp->p_nextready;
if (tmp == NIL_PROC)
        wstaw na koniec kolejki i zakończ / ustaw odpowiednia flage;
/// punkt 3
if (HEAD->NR_KOLEJKI == 1 && tmp2->NR_KOLEJKI == 2 && tmp->NR_KOLEJKI == 3)
        tmp = tmp->p_next_ready;
if (tmp == NIL_PROC)
        wstaw na koniec kolejki i zakończ / ustaw odpowiednia flage;
/// punkt 4
while(tmp != NIL_PROC && HEAD->NR_KOLEJKI >= tmp->NR_KOLEJKI)
        tmp = tmp->p_nextready;
if (tmp == NIL_PROC)
        wstaw na koniec kolejki i zakończ / ustaw odpowiednia flage;
if (tmp != NIL_PROC)
        wstaw HEAD w miejsce określone przez tmp
```

- w funkcji ready(proc*) należy zmodyfikować metodę wstawiania procesu do kolejki USER_Q. Oto moja propozycja algorytmu modyfikującego ten fragment kodu:

Niech proces wstawiany do kolejki nazywa się WSTAWIANY

1) szukam pierwszego procesu z numerem kolejki większym od numeru kolejki procesu WSTAWIANY zaczynając od pierwszej pozycji w kolejce, niech proces ten będzie wskazywany przez pozycję POZYCJA;
- jeśli taki proces nie istnieje, to proces WSTAWIANY wstawiany jest na koniec kolejki;

2) sprawdzam przypadek szczególny: numer kolejki WSTAWIANY wynosi 2, numer kolejki procesu znajdującego się bezpośrednio przed pozycją POZYCJA wynosi 2 oraz numer procesu znajdującego się bezpośrednio za pozycją POZYCJA wynosi 1. Wtedy kolejka procesów wygląda tak: … 2, 3, 1... (gdzie te liczby to numery kolejek tych procesów). W tym przypadku należy zamienić miejscami procesy których numer kolejki wynosi 3 i 1. Po zmianie kolejka będzie wyglądała tak: … 2, 1, 3... (należy pamiętać że POZYCJA cały czas wskazuje na numer 3).
- jeśli procesy takie nie istnieją, to proces WSTAWIANY wstawiany jest na koniec kolejki;

3) wstawiam proces WSTAWIANY na pozycję znajdującą się bezpośrednio przed pozycją POZYCJA

PSEUDOKOD

```
tmp = HEAD;
while(tmp != NIL_PROC && WSTAWIANY->NR_KOLEJKI >= tmp->NR_KOLEJKI)
        tmp = tmp->p_nextready;
if (tmp == NIL_PROC)
        wstaw na koniec i zakończ / ustaw odpowiednią flagę
if(tmp == HEAD)
        wstaw na początek, ustaw nowy HEAD i zakończ / ustaw odpowiednie flagi

if( WSTAWIANY->NR_KOLEJKI == 2 && bezpośrednio_przed_tmp->NR_KOLEJKI == 2 &&
tmp->p_nextready != NIL_PROC && tmp->p_nextready->NR_KOLEJKI = 1)
        zamień tmp z tmp->p_nextready miejscami, niech tmp wskazuje na to co wskazywało

wstaw WSTAWIANY na pozycje bezpośrednio_przed_tmp;
```

Plik usr/src/kernel/system.c:

- należy zadbać o to, aby nowy proces odpowiednio ustawił dodane pole struktury. W tym celu należy zmodyfikować funkcje do_fork(message*) tak, aby domyślnie ustawiała zmienną systemową NR_KOLEJKI na 1;

Ponadto należy zaimplementować dwa wywołania systemowe. Pierwsze z nich służyć będzie do sprawdzenia NR_KOLEJKI w której znajduje się proces o zadanym PID. Drugie stworzone będzie w celu modyfikacji zmiennej systemowe NR_KOLEJKI procesu o zadanym PID. Oba te wywołania powinny znajdować się w mikrojądrze, jednocześnie będąc wywoływane za pośrednictwem serwera MM lub FS. W moim przypadku będzie to serwer MM.
W celu dodania wywołań dla serwera MM należy zmodyfikować następujące pliki:

Plik usr/include/minix/callnr.h:

- dodany zostanie identyfikator nowego wywołania systemowego GETPRI;
- dodany zostanie identyfikator nowego wywołania systemowego SETPRI;
- stała N_CALLS zostanie zwiększona o 2;

Plik usr/src/mm/proto.h:

- umieszczony zostanie prototyp funkcji do_getpri();
- umieszczony zostanie prototyp funkcji do_setpri();

Plik usr/src/mm/table.c:

- w odpowiednim miejscu umieszczona zostanie nazwa funkcji do_getpri();
- w odpowiednim miejscu umieszczona zostanie nazwa funkcji do_setpri();

Plik usr/src/fs/table.c:

- w analogicznym dla funkcji do_getpri() miejscu umieszczony zostanie adres pustej funkcji no_sys;
- w analogicznym dla funkcji do_setpri() miejscu umieszczony zostanie adres puste funkcji no_sys;

Plik /usr/src/mm/main.c:

- umieszczona zostanie treść wywołania do_getptr();
- umieszczona zstanie treść wywołania do_setpri();

Ponieważ odwołanie ma nastąpić do samego mikrojądra, to przykładowa postać funkcji do_setpri() będzie wyglądać następująco:

```
PUBLIC void do_setpri()
{
        message m;
        m = mm_in;
        _taskcall(SYSTASK, SYS_SETPRI, &m);
}
```

Następnie należy dodać wywołanie do mikroądra które wykona właściwe operacje na zmiennej systemowej NR_KOLEJKI, czyli albo zwróci jej wartość albo ją zmodyfikuje.
Aby to zrobić należy dokonać modyfikacji następujących plików:

Plik usr/src/include/minix/com.h:

- w sekcji SYSTASK należy dodać kod wywołania SYS_GETPRI;
- w sekcji SYSTASK należy dodać kod wywołania SYS_SETPRI;

Plik usr/src/kernel/system.c:

- należy zdefiniować prototyp funkcji do_getpri();
- należy zdefiniować prototyp funkcji do_setpri();
- w funkcji sys_task() w instrukcji switch dodać należy warunek SYS_GETPRI;
- w funkcji sys_task() w instrukcji switch dodać należy warunek SYS_SETPRI;
- należy umieścić właściwą treść wywołania do_getpri();
- należy umieścić właściwą treść wywołania do_setpri();

Proponowany algorytm do modyfikowania zmiennej systemowej NR_KOLEJKI konkretnego procesu:

Niech element w którym dokonujemy modyfikacji to ZMIENIANY

1) w zmiennej tmp zapamiętany zostanie wskaźnik na element przed ZMIENIANYM a w zmiennej tmp2 wskaźnik na element po ZMIENIANY;

2) element ZMIENIANY zostanie wyjęty z kolejki;

3) kolejka będzie przeglądana począwszy od tmp2 w celu znalezienia kolejnego elementu o takim samym numerze kolejki jak numer kolejki elementu ZMIENIANY;

4) jeśli element nie zostanie znaleziony, nastąpi połączenie kolejki w sposób: tmp-> nast. = tmp2 a następnie przejście do punktu 5;

jeśli element zostanie znaleziony, wskaźnik na element przed nim zostanie zapamiętany w zmiennej tmp3 a wskaźnik na element po nim w zmiennej tmp4. Następnie element ten zostanie wyjęty z kolejki (niech nazywa się WYJETY) i przeprowadzone zostaną operacje typu:
tmp->nast = WYJETY; WYJETY->nast = tmp2;
tmp = tmp3; tmp2 = tmp4; oraz nastąpi powrót do punktu 4.

5) zostanie zmodyfikowana zmienna systemowa NR_KOLEJKI elementu ZMIENIANY;

6) element zostanie wstawiony do kolejki zgodnie z algorytmem przedstawionym dla funkcji ready()

4. Opis programów testujących:

Jeśli chodzi o tworzenie procesów do funkcji testującej to rozpatruje dwie opcje:
Pierwsza zakłada, że za pomocą funkcji fork() stworze określoną liczbę procesów z domyślnym zmienną systemową NR_KOLEJKI równą 1. Następnie zmienna systemowa NR_KOLEJKI zostanie zmieniona dla pewnych określonych procesów.
Druga zakłada odpalenie skryptem działających w tle kilku instancji danego programu (napisanych przeze mnie w C), a następnie modyfikacja ich zmiennych systemowych NR_KOLEJKI.

Aby umożliwić testowanie zaimplementowanych algorytmów można w funkcji sched() umieścić operacje wyprowadzania na ekran takich informacji jak: numer procesu/ PID (coś co by identyfikowało proces) oraz NR_KOLEJKI procesu znajdującego się na początku kolejki. Analogiczny rezultat można osiągnąć poprzez wykorzystanie stworzonego wcześniej wywołania systemowego do_getpri() przy znajomości wartości PID danego procesu.
Aby pokazać, że czas wykonywania kilku instancji tego samego programu jest różny w przypadku gdy znajdują się one w różnych 'abstrakcyjnych' kolejkach (kilka instancji danego programu działających w tle odpaliłbym w skrypcie, a także zmodyfikował tam numery kolejek części z nich), po zakończeniu wykonywania należy wyświetlić na ekranie numer identyfikujący proces, jego numer w kolejce oraz całkowity czas wykonania.