Warszawa, 18.11.2019

student: Patryk Jan Sozański
grupa: 2I5
nr albumu: 300258

SYSTEMY OPERACYJNE: LABORATORIUM NR 2

RAPORT Z WYKONANEGO ĆWICZENIA

Treść zadania

Zrealizować algorytm szeregowania dzielący procesy użytkownika na grupy: A, B.

Proces jest umieszczony w grupie A, gdy jego identyfikator procesu jest podzielny bez reszty przez 2 i B, gdy jego identyfikator procesu nie jest podzielny bez reszty przez 2.

Wykonać niezbędne modyfikacje funkcji systemowych umożliwiającą przenoszenie procesów pomiędzy powyższymi grupami.

Proszę także wykonać usługę systemową o prototypie: int set_scheduler(int x); która ustali proporcje czasowe w jakich scheduler ma wybierać zadania A i B do wykonania. Argument X może przyjmować wartości 0..100, i oznacza to ile procent czasu dostanie zadanie A (zadanie B dostanie odpowiednio 100-X procent czasu). Usługa ma zwracać 0 gdy udało się wykonać zmianę proporcji czasowych, gdy zwróci -1, oznaczać będzie proces został przydzielony do klasy B i procesom tej klasy nie wolno zmieniać proporcji.

Opracować również łatwą metodę weryfikacji poprawności rozwiązania.

Rozwiązanie

W celu rozwiązania zadania wykonałem następujące czynności:

- w pliku /usr/src/kernel/proc.h zmodyfikowałem strukturę proc, dodając nowe pole na końcu:

  int group;

  które w zależności od PID procesu, przyjmuje jedną ze zdefiniowanych w tym samym pliku wartości:

  #define GROUPA 0
  #define GROUPB 1
  #define DEFAULTGROUP -1

  w tym samym pliku umieściłem deklaracje dwóch zmiennych służących do ustalania proporcji czasowych między długościami kwantów dla poszczególnych grup procesów:

  EXTERN int quantum_percent;
  EXTERN int current_quantum;

- w pliku /usr/src/kernel/proc.c zmodyfikowałem funkcję pick_proc, tak aby w zależności od wartości zmiennej quantum_percent i przynależności procesu do danej grupy, nadawała różne wartości zmiennej current_quantum;

  current_quantum = (rp -> group == GROUPA) ? (int)((100 - quantum_percent)/100) : (int)(quantum_percent/100);

- w pliku /usr/src/kernel/clock.c zmodyfikowałem funkcję do_clocktick, aby w odpowiednim momencie wywłaszczała procesy w zależności od przynależności procesów do grupy A lub B;

- w pliku /usr/src/kernel/system.c dodałem obsługę czterech funkcji systemowych o nazwach:

  PRIVATE int do_setsched(m_ptr);
  - zmienia wartość zmiennej quantum_percent na zadaną;

  PRIVATE int do_getsched(m_ptr);
  - zwraca wartość zmiennej quantum_percent;

  PRIVATE int do_setgroup(m_ptr);
  - zmienia wartość pola group procesu o podanym PID na zadaną;

  PRIVATE int do_getgroup(m_ptr);
  - zwraca wartość pola group procesu o podanym PID;

  w tym samym pliku rozszerzyłem funkcję do_fork o algorytm przyporządkowujący proces do grupy:

```
if(rpc -> p_pid % 2 == 0)
        rpc -> group = GROUPA;
 else if(rpc -> p_pid % 2 == 1)
        rpc -> group = GROUPB;
```

- w pliku /usr/src/kernel/main.c nadałem zmiennej quantum_percent wartość początkową równą 50;

- w pliku /usr/include/minix/callnr.h zwiększyłem stałą N_CALLS i dodałem na końcu identyfikator nowych wywołań systemowych;

- w pliku /usr/include/minix/com.h zdefiniowałem stałe do obsługi funkcji systemowych;

- w pliku /usr/src/fs/table.c nadałem wartość no_sys odpowiednim elementom tablicy;

- w pliku /usr/src/mm/table.c wstawiłem w odpowiednie miejsca tablicy adresy funkcji systemowych;

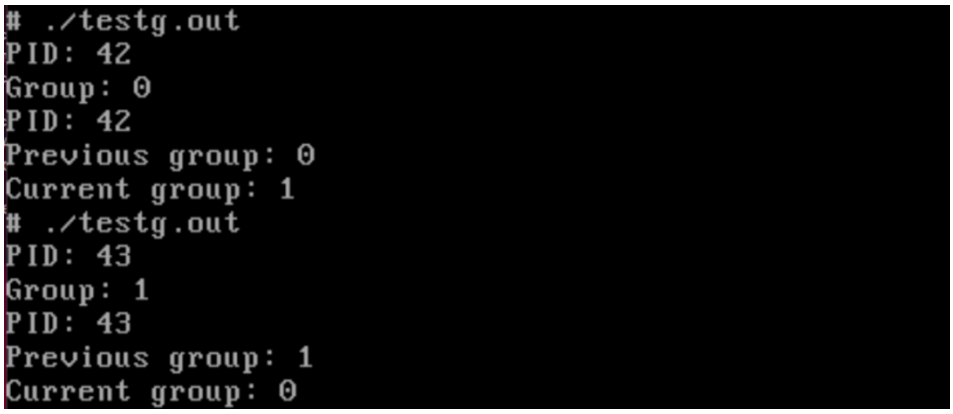
- w pliku /usr/src/mm/proto.h umieściłem prototypy funkcji systemowych;

- w pliku /usr/src/mm/main.c zamieściłem procedury obsługi funkcji systemowych;

Testowanie

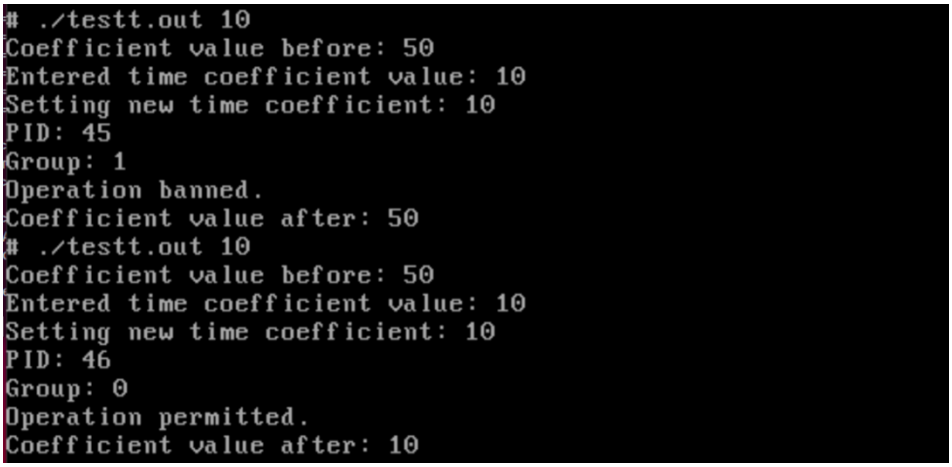
W celu sprawdzenia poprawności działania funkcji systemowych utworzyłem trzy pliki testowe:

- w pliku testgroup.c umieściłem algorytm sprawdzający poprawność przydzielania procesów do grup oraz funkcji zmieniającej grupę procesu:



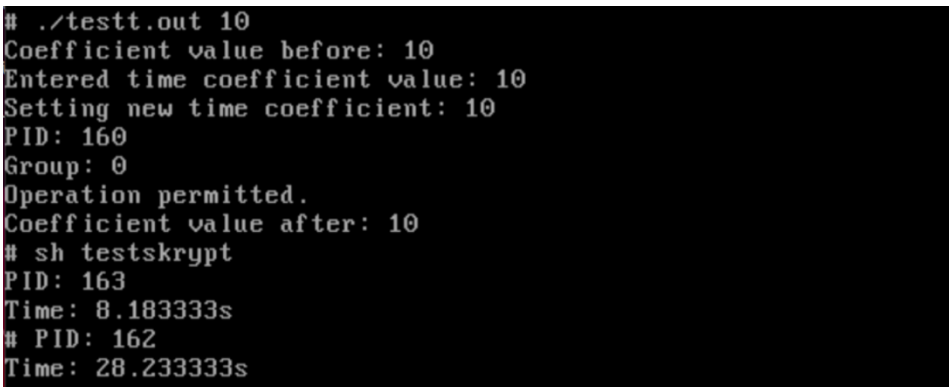
Program testujący odczytuje wartość PID procesu, następnie wywołuje funkcję systemową odczytującą grupę, do jakiej należy proces, a na końcu w zależności od tego, w jakiej grupie jest proces, zamienia jego przyporządkowanie na przeciwne wywołując odpowiednia funkcję systemową.

- w pliku testtime.c umieściłem algorytm sprawdzający poprawność działania funkcji zmieniającej wartość parametru quantum_percent:

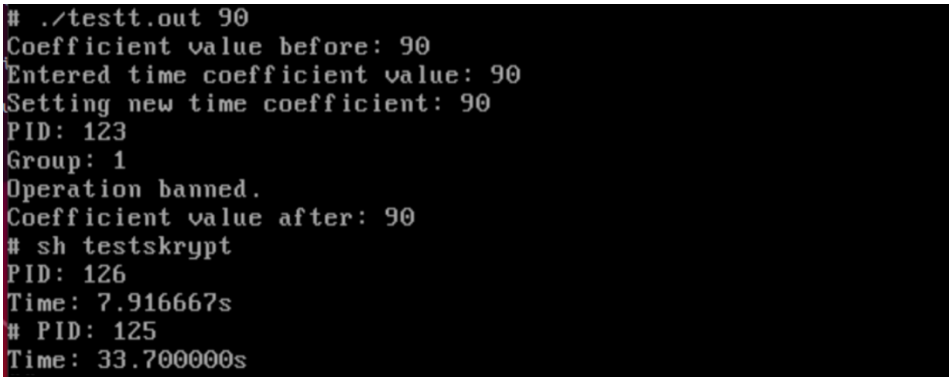


Program testujący wywołuje funkcję systemową odczytującą wartości zmiennej quantum_percent, a następnie przypisującą jej wartość podaną przez użytkownika tylko wtedy, gdy proces należy do grupy A (0).

- w pliku maintest.c umieściłem algorytm sprawdzający poprawność działania funkcji odpowiedzialnych za przydzielanie procesom odpowiedniego kwantu czasu. W tym celu mierzyłem czasy wykonywania się dwóch współbieżnych procesów należących do oddzielnych grup i porównywałem wyniki:



Dla wartości quantum_percent = 10 proces A (PID = 162) wykonywał się zauważalnie dłużej od procesu B (PID = 163).



Dla wartości quantum_percent = 90 proces A (PID = 126) wykonał się znacznie szybciej od procesu B (PID = 125).

Pomiary

wartość zmiennej quantum_percent	czas wykonania procesu z gr. A [s]	Czas wykonania procesu z gr. B [s]
10	**28,23**	**8,18**
20	**26,76**	**11,50**
30	**24,02**	**14,10**
40	**22,44**	**18,96**
50	**21,20**	**22,02**
60	**18,62**	**23,90**
70	**14,74**	**26,78**
80	**10,90**	**27,17**
90	**7,92**	**33,70**

Wnioski

Zmiana wartość zmiennej quantum_percent widocznie wpływa na czas wykonywania się dwóch współbieżnych procesów. Im wyższa wartość tej zmiennej, tym szybciej wykonuje się proces grupy A i tym wolniej wykonuje się proces grupy B. Odwrotną sytuację można zaobserwować dla coraz mniejszych wartości zmiennej.

Niedokładność pomiarów może wynikać z następujących powodów:

- w pliku clock.c makro SHED_RATE przyjmuje dyskretne wartości, przez co zmienna sched_ticks przyjmuje jedynie wartości przybliżone;
- w przypadku, gdy jeden z procesów zakończy się jako pierwszy, drugi otrzymuje cały czas procesora;
- procesy uruchamiają się z pewnym opóźnieniem względem siebie;
- płynności działania systemu minix;

Wyniki powyższych testów są zgodne z teorią i moimi przewidywaniami. Na ich podstawie stwierdzam, że wykonane przeze mnie funkcje systemowe działają poprawnie.