Szeregowanie procesów

Jakub Robaczewski

Wywołania systemowe

W moim rozwiązaniu stworzyłem 5 nowych wywołań systemowych:

- `set_group()` – ustawia grupę dla procesu o danym PID
- `get_group()` – pobiera grupę dla procesu o danym PID
- `set_schedule_a()` – ustawia % czasu procesora dla grupy A
- `set_schedule_b()` – ustawia % czasu procesora dla grupy B
- `get_time()` – pobiera % czasu procesora dla danej grupy

Wymagało to wprowadzania edycji w poniższych plikach:
`/usr/include/minix/callnr.h` -> Definicje odwołań
`/usr/src/mm/proto.h` -> Prototypy funkcji
`/usr/src/mm/table.c` -> Odwołanie do funkcji
`/usr/src/fs/table.c` -> Adresy pusty
`/usr/src/tools/main.c` -> procedury obsługi

Oprócz wywołań w folderach mm i fs, należało dodać wywołania w kernelu, ponieważ tylko one mają dostęp do funkcji szeregujących procesy. Wprowadziłem zmiany w następujących plikach.
`/usr/include/minix/com.h` -> Definicje odwołań
`/usr/src/kernel/system.c` -> Procedury obsługi i prototypy

Szeregowanie

By zrealizować odpowiednie szeregowanie, dodałem zmienne `group` i `iterator` do struktury proc w pliku `/usr/src/kernel/proc.h`, oraz ustawiłem ich domyślne wartości w funkcji `do_fork()` w pliku `/usr/src/kernel/proc.c`.

Zmienne globalne określające % czasu procesora dla każdej z grup zapisałem w tablicy `group_times`, która zainicjowałem w `proc.c`.

Algorytm szeregowania

Główny algorytm szeregujący znajduje się w pliku `/usr/src/kernel/proc.c` i wykonuje się w oparciu o następujące założenia.

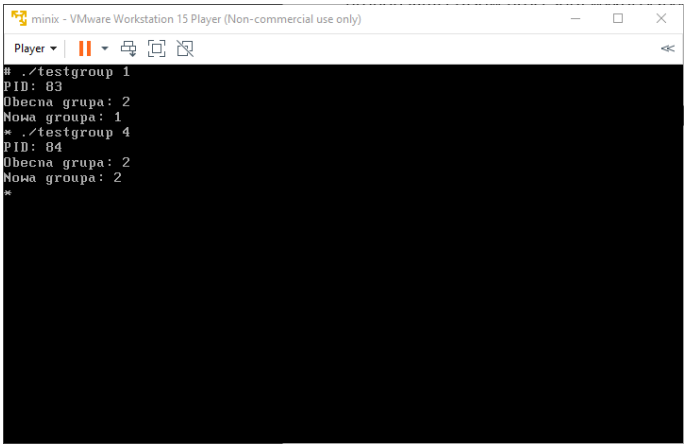
- Za każdym razem gdy dany proces wywołuje sched() (około 100 ms), algorytm zwiększa iterator procesu.
- Następnie odczytywana jest wartość % czasu procesora dla jego grupy i porównywana z iteratorem.
- Gdy iterator ją przekroczy jest on zerowany, a algorytm wybiera następny proces z kolejki.
- Algorytm przegląda kolejkę tak długo, aż znajdzie proces z grupy o niezerowym czasie

```
PRIVATE void sched()
{
/* The current process has run too long.  If another low priority (user)
 * process is runnable, put the current process on the end of the user queue,
 * possibly promoting another user to head of the queue.
 */

  struct proc *c_proc, *n_proc;
  int time, n_group, i, control=1;

  if (rdy_head[USER_Q] == NIL_PROC) return;

  c_proc = rdy_head[USER_Q];
  c_proc->iterator += 1;
  n_group = c_proc->group;
  time = time_group[n_group];

  if (c_proc->iterator < time) {
     pick_proc();
     return;
  }
  c_proc->iterator = 0;

  while (control == 1) {
        /* One or more user processes queued. */
        rdy_tail[USER_Q]->p_nextready = rdy_head[USER_Q];
        rdy_tail[USER_Q] = rdy_head[USER_Q];
        rdy_head[USER_Q] = rdy_head[USER_Q]->p_nextready;
        rdy_tail[USER_Q]->p_nextready = NIL_PROC;

        n_group = rdy_head[USER_Q]->group;
        time = time_group[n_group];

        if (time != 0) {
             pick_proc();
            return;
        }
  }
}
```

Testowanie

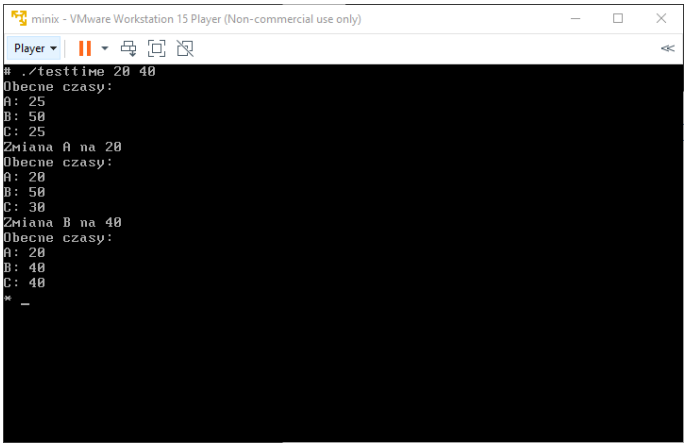
Wywołania systemowe są testowane za pomocą programów `/root/testgroup` i `/root/testtime`, natomiast algorytm testujący jest testowany za pomocą skryptu `/root/test.sh`, który jednocześnie uruchamia 3 programy `/root/longtest` nadając im kolejno grupy A, B i C. Programy na końcu swojego działania wypisują czas mierzony bezwzględnie i względnie oraz ich różnicę (czas oczekiwania). Proporcje między tymi czasami powinny być zgodne z proporcjami czasów procesora wykorzystywanych w różnych grupach.

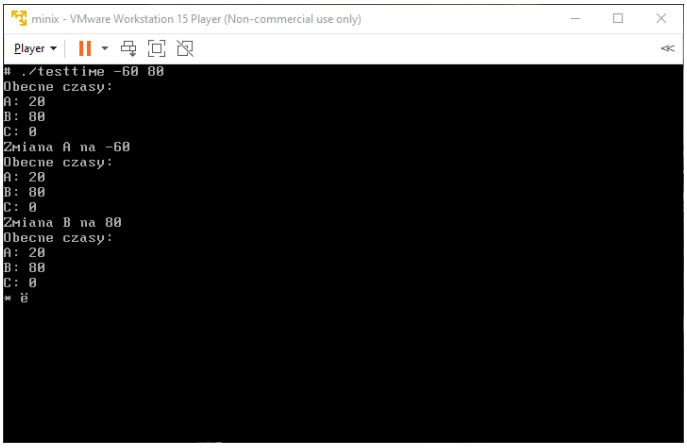
./testgroup [new group]



Program uruchamia się podając numer nowej grupy. Program ustawia swoją grupę na podaną. W przypadku podania błędnych danych grupa nie zostanie ustawiona.

./testtime [time for A] [time for B]

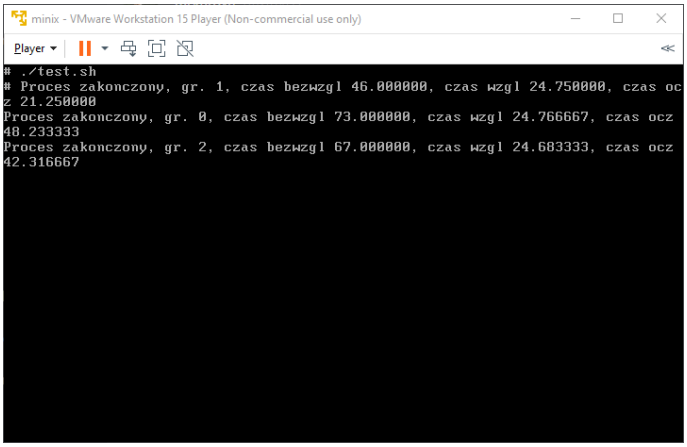




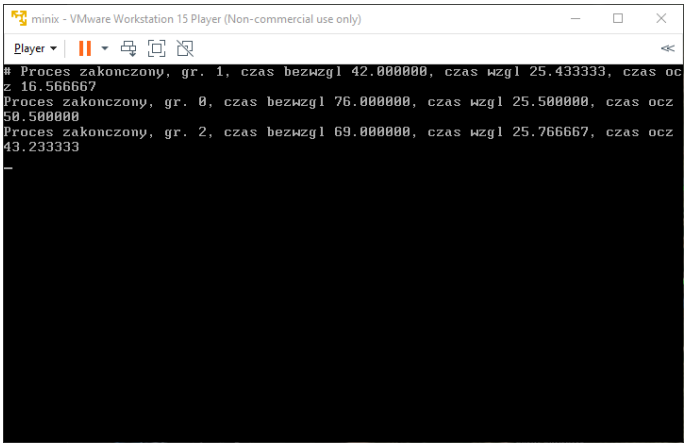
Program ustawia zmienne globalne dla grupy a i grupy b (oraz niejawnie dla grupy c, ponieważ c = 100 – a – b). Podobnie jak w poprzednim przypadku podanie błędnych wartości sprawi, że dane nie zostaną ustawione poprawnie.

./test.sh

Głównym program testujący, uruchamia program ./longtest w 3 różnych grupach, którywykonuje obliczenia i mierzy czas bezwzględny, względny oraz różnicę (czas oczekiwania).



A=25, B=50, C=25



A=20, B=60, C=20

Zauważamy, że dla czasów 25, 50, 25 czas oczekiwania zachowuje podobne proporcje. Grupa B, która wykonywała się 2 razy częściej oczekuje 2 razy krócej niż pozostałe. Podobne właściwości zauważamy dla testu 20, 60, 20.