Dokumentacja zadanie 2

Cel zadania

Zmodyfikować algorytm szeregowania procesów użytkownika w systemie Minix. Procesy użytkownika mają zostać przydzielone do jednej z 3 grup: A, B, C.

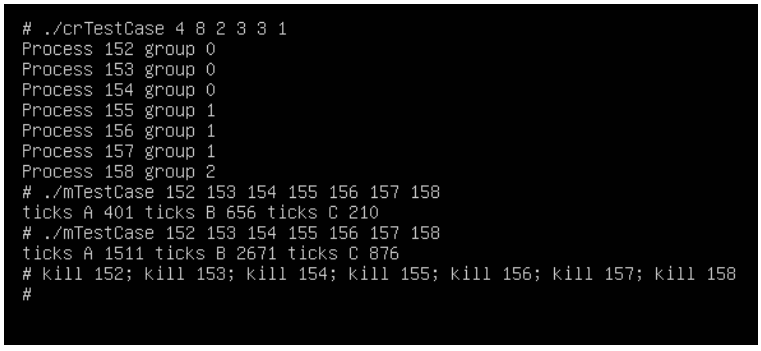
Powinny cyklicznie wykonywać się procesy z grup A,B,C, oraz cyklicznie procesy wewnątrz danej grupy.

Używanie funkcji testujących

Tworzenie przypadku testowego - **crTestCase**

Podajemy liczby kwantów dla kolejnych grup i liczbę procesów w danych grupach. Program ustawia odpowiednio kwanty oraz tworzy procesy z danych grup, działające w pętlach nieskończonych.

./crTestCase quantsA quantsB quantsC #procA #procB #procC



Odczytanie liczby tcków (user_time) jaka została przydzielona w sumie procesom z danej grupy- **mTestCase**

Jako argumenty podajemy pidy procesów, które chcemy brać pod uwagę (docelowo: pidy wszystkich utworzonych przez crTestCase procesów)

./mTestCase pid1 pid2 pid3 pid4 …

Na koniec: zabicie wszystkich procesów z przypadku testowego (kill pid)

1. Wymyślenie szeregowania, napisanie funkcji sched

Rozważenie przypadków:

- w kolejce nie ma żadnych procesów
- jest 1 proces, 2 procesy
- jest 3 lub więcej procesów
- nie ma żadnego procesu z którejś z grup
- są tylko procesy z jednej z grup
- znaleziony proces jest drugi/w środku/na końcu kolejki

```
void sched() {
    // 0 processes in queue -> leave as it is
    if (rdy_head == NULL) return;
    // 1 process in queue -> leave as it is
    if (rdy_head == rdy_tail) return;
    // 2 processes in queue -> swap order
    if (rdy_head->p_nextready == rdy_tail) {
        change_order_simple_queue(); return;
    }

    // 3+ processes in queue
    find_next_process();
    if (parent_node == rdy_head) {
        change_order_simple_queue();
//        printf("second\n");
    } else if (parent_node->p_nextready == rdy_tail) {
        change_order_found_last();
//        printf("last\n");
    } else {
        change_order_found_middle();
//        printf("middle\n");
    }
}
```

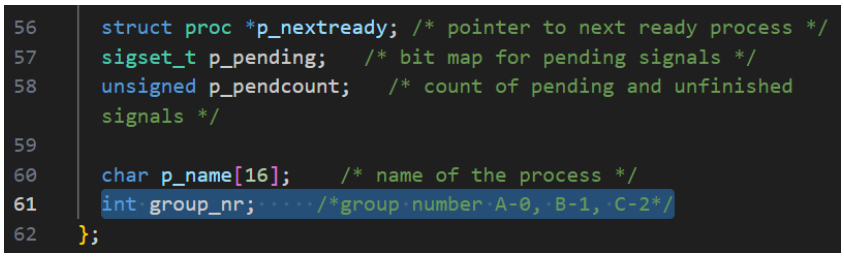
```
void change_order_simple_queue() {
    // Normal sched behaviour
    rdy_tail->p_nextready = rdy_head;
    rdy_tail = rdy_head;
    rdy_head = rdy_head->p_nextready;
    rdy_tail->p_nextready = NULL;
}

void change_order_found_middle() {
    // Moving previous head to the end of queue
    rdy_tail->p_nextready = rdy_head;
    // Setting found next process as the new head
    rdy_head = parent_node->p_nextready;
    // Connecting nodes before and after found process
    parent_node->p_nextready = parent_node->p_nextready->p_nextready;
    // Connecting previous head children to new head (rdy_tail->p_nextready->p_nextready = old head child)
    rdy_head->p_nextready = rdy_tail->p_nextready->p_nextready;
    // Setting old head=new tail child to none
    rdy_tail->p_nextready->p_nextready = NULL;
```

```
    // Setting old head as new tail
    rdy_tail = rdy_tail->p_nextready;
}

void change_order_found_last() {
    // Moving old head to the end
    parent_node->p_nextready = rdy_head;
    // New head takes over old head's children
    rdy_tail->p_nextready = rdy_head->p_nextready;
    // Setting old head=new tail child to none
    rdy_head->p_nextready = NULL;
    // Updating head pointer
    rdy_head = rdy_tail;
    // Updating tail pointer
    rdy_tail = parent_node->p_nextready;
}

void find_next_process() {
    proc* current_process;
    // Maximum 3 tries
    for (int try_nr = 0; try_nr < 3; try_nr++ ) {
        // Determining next browsed group nr
        curr_group++;
        if (curr_group > 2)
            curr_group = 0;
        // Going through queue looking for process in group
        current_process = rdy_head;
        while (current_process->p_nextready != NULL) {
            // Found process from current group
            if (current_process->p_nextready->group == curr_group) {
                parent_node = current_process; return;
            }
            // Moving to next process
            current_process = current_process->p_nextready;
        } // Did not find process from current group, moving to next group
    }
}
```

2.Przydzielanie procesu grupy

usr/src/kernel/proc.h

dodanie pola nr grupy



usr/src/kernel/main.c

ustawiamy grupę initowi, a kazde jego dziecko jest jego kopią, więc będzie miało taką samą grupę

```
134   proc[NR_TASKS+INIT_PROC_NR].p_pid = 1;/* INIT of course has
      pid 1 */
135   proc[NR_TASKS+INIT_PROC_NR].group_nr = 0; /*setting default
      group to A*/
136   bill_ptr = proc_addr(IDLE);       /* it has to point
      somewhere */
137   proc_addr(IDLE)->p_priority = PPRI_IDLE;
138   lock_pick_proc();
```

usr/src/kernel/system.c

ustawienie domyślnego nr grupy w do_fork (powinno to byc 0)

```
235     rpc->sys_time = 0;
236     rpc->child_utime = 0;
237     rpc->child_stime = 0;
238     rpc->group_nr = 1; /*Setting default group nr*/
239
240     return(OK);
241   }
```

usr/src/kernel/dmp.c

dodanie wyświetlania pod F1 nr grupy oraz p_nr (indeksu)

```
39      printf(" %d %5lx %6lx %2x %7lu %7lu %5uK %5uK %5uK ",
40              rp->group_nr,
41             (unsigned long) rp->p_reg.pc,
42             (unsigned long) rp->p_reg.sp,
43             rp->p_flags,
44             rp->user_time, rp->sys_time,
```

3. Dodanie wywołań systemowych do wyświetlania i zmiany grupy, kwantów dla grupy

src/kernel/system.c

Prototypy i ciała funkcji obsługujących wywołania, informacja jaka funkcja obsługuje dane wywołanie

```
245  PRIVATE int do_setQuants(m_ptr)
246  register message *m_ptr;  /* pointer
     to request message */
247  {
248      /* w m1_i1 - indeks (numer) grupy
249          w m1_i2 - nowa liczba kwantow
             dla grupy */
250      printf("Group %d", m_ptr->m1_i1);
251      if ((m_ptr->m1_i1 < 0) ||
         (m_ptr->m1_i1 > 2) ||
         (m_ptr->m1_i2 < 1)) return EINVAL;
252      quants[m_ptr->m1_i1] =
         m_ptr->m1_i2;
253      printf(" new quants %d=%d",
         m_ptr->m1_i2, quants
         [m_ptr->m1_i1]);
254      return(OK);
255  }
```

include/minix/com.h

Nry wywołań kernelowych

```
1   /* System calls. */
```

src/mm/misc.c

Wywołania MM - ciało

```
17  PUBLIC int do_getgroup(){
18      int g_nr = _taskcall(SYSTASK, SYS_GETGROUP, &mm_in);
19      mp->mp_reply = mm_in;
20      return g_nr;
21  }
22
23  PUBLIC int do_setgroup(){
24      return _taskcall(SYSTASK, SYS_SETGROUP, &mm_in);
25  }
26
27  PUBLIC int do_getquants(){
28      int quants = _taskcall(SYSTASK, SYS_GETQUANTS, &mm_in);
29      mp->mp_reply = mm_in;
30      return quants;
31  }
32
33  PUBLIC int do_setquants(){
34      return _taskcall(SYSTASK, SYS_SETQUANTS, &mm_in);
35  }
```

src/mm/proto.h, src/mm/table.c, src/fs/table.c, include/minix/callnr.h

inne elementy wywolan systemowych MM

pliki testowe wywolan

```c
#include <lib.h>
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char* argv[])
{
        message m;
        int group = 0;
        group = m.m1_i1 = atoi(argv[1]);
        _syscall(MM, GETQUANTS, &m);
```

Dodawanie procesów z grupą odziedziczoną od rodzica, zmienianie ich grupy wywołaniem systemowym

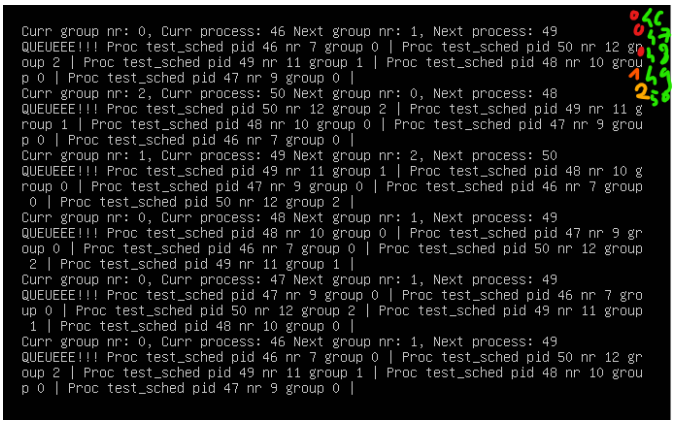
```c
int create_children = 5;
int groups[5] = {0,0,0,1,2};
```

```
# ./test_sched &
# creating 5 children
Child pid: 141 Group: 0
Child pid: 142 Group: 0
Child pid: 143 Group: 0
Child pid: 144 Group: 1
Child pid: 145 Group: 2
```

```
pid  vir gr
( -1)  -1  0      0   e94c  0   21062       0     2K    61K   119K        HARDWA
R
(  0)  0  0   3551   afc0  4      10       0  1024K  1039K    59K ANY    MM
(  1)  1  0   701d  1c7c4  4     149       0  1083K  1112K   143K ANY    FS
    1  2  1   17a5   1310  4       0       3  1226K  1233K    12K MM     INIT
  126  3  1    1e5  20254  4       0       2  1238K  1245K   136K MM     test_sc
hed
   20  4  1   eba1   f638  4       1      11   204K   267K   125K FS     sh
   15  5  1    539   12fc  4       0       4   375K   377K     7K MM     update
   16  6  1   4019   9b04  4       0       2   382K   405K    62K MM     cron
  127  7  0    1e5  20254  4       0       0  1238K   444K   136K MM     test_sc
hed
   21  8  1   1671   41b8  4       0       1   126K   149K    23K FS     getty
  128  9  0    1e5  20254  4       0       0  1238K  1374K   136K MM     test_sc
hed
--more--
--pid --p_nr --gnr --pc- ---sp- flag -user --sys-- -text- -data- -size- -recv- c
ommand
  129  10  0    1e5  20254  4       0       0  1238K  1503K   136K MM     test_s
ched
  130  11  1    1e5  20254  4       0       0  1238K  1632K   136K MM     test_s
ched
  131  12  2    1e5  20254  4       0       0  1238K  1761K   136K MM     test_s
ched
```

4. Zastąpienie algorytmu szeregowania (sched)

Test znajdywania kolejnego procesu

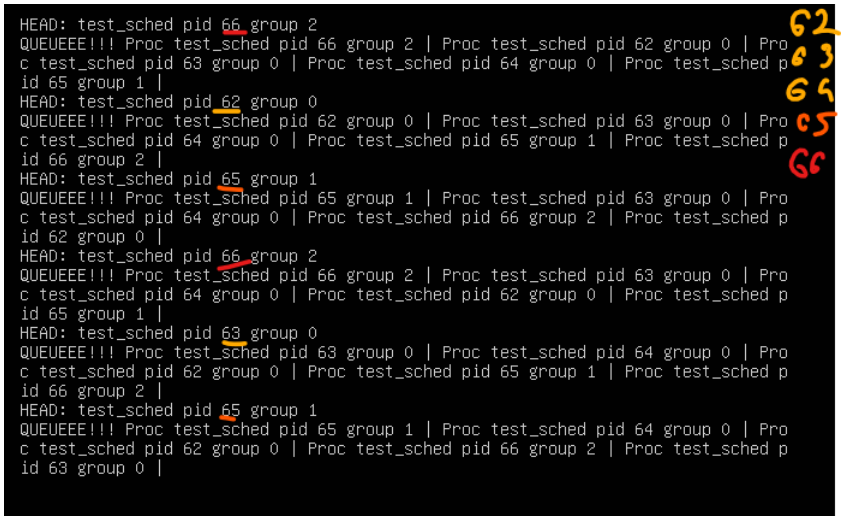


kernel/proc.c

Dodanie napisanego wcześniej sched, dla roznych przypadkow, z pomocnicza funkcja znajdowania kolejnego procesu z odpowiedniej grupy

```
530     /* 1 process in queue -> leave as it is */
531     if (rdy_head[USER_Q] == rdy_tail[USER_Q]) {
532       pick_proc(); return;
533     }
534     /* 2 processes in queue -> swap order */
535     if (rdy_head[USER_Q]->p_nextready == rdy_tail[USER_Q]) {
536         change_order_simple_queue(); pick_proc(); return;
537     }
538
539     /* 3+ processes in queue */
540     find_next_process();
541     if (parent_node == rdy_head[USER_Q]) {
542         change_order_simple_queue();
543     }
544     else if (parent_node->p_nextready == rdy_tail[USER_Q]) {
545         change_order_found_last();
546     }
547     else {
548         change_order_found_middle();
549     }
550     pick_proc();
551   }
```

Sprawdzanie konkretnie jak wygląda kolejka ustawiana wg sched - działa



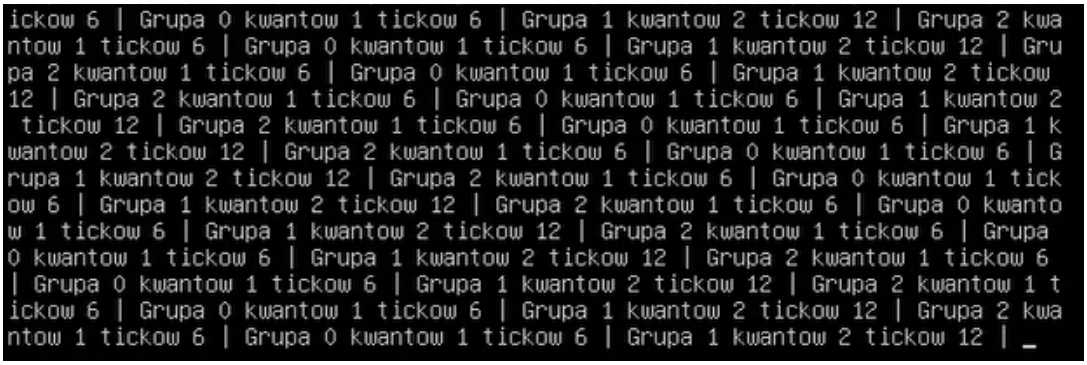
5. Testy

Tabela z testami dla różnych parametrów

https://wutwaw-my.sharepoint.com/:x:/g/personal/01178572_pw_edu_pl/EQOHLnHNLLRIoSSk9TjdNe8BnvXdKkVxbn-D3eB9Cj4EGw?e=w8H0N9

Ustawianie procesów w odpowiedniej kolejności (przy 1 procesie z danej grupy)✅

Poprawna informacja ile kwantów/ticków powinien dostać dany proces w zależności od grupy ✅



Rzeczywiście dodawanie poprawnej liczby ticków ❌

```
Process 36 group 2
#  = time 12 | Time 12 + 12 tickow = time 30 | Time 30 + 12 tickow = time 48 | T
ime 48 + 12 tickow = time 66 | Time 66 + 12 tickow = time 84 | Time 84 + 12 tick
ow = time 102 | Time 102 + 12 tickow = time 120 | Time 120 + 12 tickow = time 13
8 | Time 138 + 12 tickow = time 156 | Time 156 + 12 tickow = time 174 | Time 174
 + 12 tickow = time 192 | Time 192 + 12 tickow_
```

Rzeczywista przydzielana procesowi liczba ticków ZA KAŻDYM RAZEM jest większa o 6 (większa o 1 kwant czasu) - niezależnie od tego ile kwantów dany proces miał otrzymać, zawsze będzie o to 1 więcej → zaburza to testy, ale pomijając to zjawisko, wyniki są poprawne

(wyświetlany czas to czas procesu z grupy 1)

```
# ./crTestCase 1 1 1 1 1 1
Process 41 group 0
Process 42 group 1
Process 43 group 2
#  = time 12 | Time 12 + 6 tickow = time 24 | Time 24 + 6 tickow = time 36 | Tim
e 36 + 6 tickow = time 48 | Time 48 + 6 tickow = time 60 | Time 60 + 6 tickow =
time 72 | Time 72 + 6 tickow = time 84 | Time 84 + 6 tickow = time 96 | Time 96
+ 6 tickow = time 108 | Time 108 + 6 tickow
```

```
# ./crTestCase 2 1 1 1 1 1
Process 48 group 0
Process 49 group 1
Process 50 group 2
#  = time 18 | Time 18 + 6 tickow = time 30 | Time 30 + 6 tickow = time 42 | Tim
e 42 + 6 tickow = time 54 | Time 54 + 6 tickow = time 66 | Time 66 + 6 tickow =
time 78 | Time 78 + 6 tickow = time 90 | Time 90 + 6 tickow = time 102 | Time 10
2 + 6 tickow = time 114 | Time 114 + 6 tickow_
```

```
# ./crTestCase 1 2 1 1 1 1
Process 55 group 0
Process 56 group 1
Process 57 group 2
#  = time 12 | Time 12 + 12 tickow = time 30 | Time 30 + 12 tickow = time 48 | T
ime 48 + 12 tickow = time 66 | Time 66 + 12 tickow = time 84 | Time 84 + 12 tick
ow = time 102 | Time 102 + 12 tickow = time 120 | Time 120 + 12 tickow = time 13
8 | Time 138 + 12 tickow
```

Dalsze testy 🔄

```
# ./crTestCase 1 1 1 2 1 1
Process 57 group 0
Process 58 group 0
Process 59 group 1
Process 60 group 2
# ./mTestCase 57 58 59 60
ticks A 223 ticks B 204 ticks C 204
# ./mTestCase 57 58 59 60
ticks A 252 ticks B 240 ticks C 240
# ./mTestCase 57 58 59 60
ticks A 268 ticks B 252 ticks C 252
# _
```

Pełny przypadek testowy

```
# ./crTestCase 4 8 2 3 3 1
Process 152 group 0
Process 153 group 0
Process 154 group 0
Process 155 group 1
Process 156 group 1
Process 157 group 1
Process 158 group 2
# ./mTestCase 152 153 154 155 156 157 158
ticks A 401 ticks B 656 ticks C 210
# ./mTestCase 152 153 154 155 156 157 158
ticks A 1511 ticks B 2671 ticks C 876
# kill 152; kill 153; kill 154; kill 155; kill 156; kill 157; kill 158
#
```

Test case 14	quants A	quants B	quants C	#proc A	#proc B	#proc C	ticks start A	ticks start B	ticks start C	ticks end A	ticks end B	ticks end C
	4	8	2	3	3	1	401	656	210	1511	2671	876
Results	tick diff A	tick diff B	tick diff C	time diff [ticks]	A/total ratio	B/total ratio	C/total ratio	A/B ratio	A/C ratio	B/C ratio		
	1110	2015	666	3791	0,293	0,532	0,176	0,551	0,6	0,331		

Result	Correct	Result	Correct	Result	Correct
A+1/B+1 = 5/9	A/B = 4/8	A+1/C+1 = 3/5	A/C = 2/4	B+1/C+1 = 3/9	B/C = 2/8
0,555555556	0,5	0,6	0,25	0,333333333	0,25