Отчет по курсу “Операционные системы” Задание 3.01

Владимир Руцкий, 4057/2

24 декабря 2009 г.

Постановка задачи

*“Напишите программу, допускающую одновременную работу не более трех своих экземпляров. Программа должна работать достаточно долго для того, чтобы можно было убедиться в том, что более трех копий в системе не работают (можно использовать **sleep**).”*

Выбранный метод решения

Для межпроцессорного взаимодействия используются System V семафоры.

Ключ для идентификации семафора строится по имени выполняемой программы (**ftok**), что гарантирует, что семафор будет один для всех запущенных копий программы.

Семантика значения семафора: сколько ещё копий программы можно запустить, параллельно с уже запущенными.

1. При запуске программы выполняется попытка создания нового семафора (**semget**). В случае успеха будет получен идентификатор нового семафора, и выполняется его инициализация максимальным количеством одновременно запускаемых копий программ (**semctl**). Если семафор уже был создан, выполняется получение его идентификатора.

2. После получения идентификатора происходит попытка атомарного уменьшения значения семафора (**semop**). Если попытка удалась — программа работает дальше, иначе количество запущенных копий уже достигло предела, и программа завершает работу.

   Вызов атомарного уменьшения семаформа производится с параметрами, обеспечивающими откат операции уменьшения, при завершении программы, это гарантирует, что выделенный на копию запущенной программы “ресурс” будет освобожден по завершении работы программы.

Исходный код

Исходный код 1: task_3_01.c

```
1  /* task_3_01.c
2   * Task 3.01 on Unix course.
3   * Vladimir Rutsky <altsysrq@gmail.com>
4   * 24.12.2009
5   */
6
7  /* Program allows run not more than 3 copy at the same time
8   * using System V semaphores.
9   */
10
11 #include <stddef.h>
12 #include <stdio.h>
13 #include <stdlib.h>
14 #include <string.h>
15 #include <errno.h>
16
17 #include <sys/types.h>
18 #include <sys/ipc.h>
19 #include <sys/sem.h>
20
21 static int const maxNProcesses = 3;
22
23 int tryRun( key_t key )
24 {
25   int semId;
26
27   /* Semaphor already exists.
28    * Locating and using it */
29   if ((semId = semget(key, 0, IPC_CREAT)) != -1)
30   {
31     struct sembuf sops[1];
32     int semopResult;
33
34     sops[0].sem_num =  0;          /* Operate on semaphore 0 */
35     sops[0].sem_op  = -1;          /* Decrement semaphor value */
36     sops[0].sem_flg =  IPC_NOWAIT | SEM_UNDO;
37                                    /* Don't wait for free semaphor
                                          resources,
38                                     * and free resource automatically on
39                                     * process exit */
40
41     /* Decreasing semaphor on 1 if possible.
42      * If not possible - quiting */
43     if ((semopResult = semop(semId, sops, 1)) != -1)
44     {
45       /* Allocated resource.
46        * Waiting current process termination to free resource */
47       /* TODO: If program will be terminated here, resource will not be
             freed */
48       fprintf(stdout, "Program is running.\n");
49
50       {
```

```
51          int instancesLeft;
52          if ((instancesLeft = semctl(semId, 0, GETVAL)) != -1)
53          {
54            fprintf(stdout, "%d instances left.\n", instancesLeft);
55          }
56          else
57          {
58            perror("semctl");
59            fprintf(stderr, "Error: semctl.\n");
60          }
61        }
62        fprintf(stdout, "Press enter to terminate...\n");
63        fgetc(stdin);
64      }
65      else if (errno == EAGAIN)
66      {
67        /* No free resources. Quiting */
68        fprintf(stdout, "All %d program instances is running.\n",
              maxNProcesses);
69        return 0;
70      }
71      else
72      {
73        perror("semop");
74        fprintf(stderr, "Error: semop() failed.\n");
75        return -1;
76      }
77    }
78    else
79    {
80      perror("semget");
81      fprintf(stderr, "Error: semget() failed.\n");
82      return -1;
83    }
84
85    return 0;
86  }
87
88  int main( int argc, char const *argv[] )
89  {
90    char const *keyPathName = argv[0];
91    int const keyProjId = 1;
92
93    key_t key;
94
95    /* All program processes will share same IPC key created
96     * from program name and project id equal to 1. */
97    if ((key = ftok(keyPathName, keyProjId)) != -1)
98    {
99      int semId;
100
101     /* Trying to create new semaphor */
102     if ((semId = semget(key, 1, IPC_CREAT | IPC_EXCL | 0600)) != -1)
103     {
104       /* New not initialized semaphor created.
105        * Initializing it */
```

```
106
107         if (semctl(semId, 0, SETVAL, maxNProcesses) != -1)
108         {
109           /* Initialized.
110            * Running */
111           int result;
112           if ((result = tryRun(key)) != 0)
113             return result;
114         }
115         else
116         {
117           perror("semctl");
118           fprintf(stderr, "Error: semctl() failed.\n");
119         }
120       }
121       else if (errno == EEXIST)
122       {
123         int result;
124         if ((result = tryRun(key)) != 0)
125           return result;
126       }
127       else
128       {
129         /* Some actual failure in creating semaphor */
130         perror("semget");
131         fprintf(stderr, "Error: semget() failed.\n");
132         return -1;
133       }
134     }
135     else
136     {
137       perror("ftok");
138       fprintf(stderr, "Error: ftok('%s', %d) failed.\n", keyPathName,
              keyProjId);
139       return -1;
140     }
141
142     return 0;
143 }
```