Санкт-Петербургский Государственный Политехнический Университет
Кафедра «Прикладная математика»

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №4

По курсу “Операционные системы”

Выполнил студент гр. 4057/2: Зенцев Ф.К.

Преподаватель: Тимофеев Д.А.

Санкт-Петербург 2010

Формулировка задания (4.3): Напишите программу, моделирующую решение задачи «спящего брадобрея». Задачу необходимо решить с помощью потоков POSIX.

Формулировка задачи: Парикмахерская состоит из N стульев и стула, где брадобрей работает с посетителем. Если посетителей нет, то брадобрей ложится спать. Если заходит посетитель, но все стулья заняты, он уходит. Если брадобрей занят, но свободный стул есть, тогда посетитель занимает свободный стул. Если же брадобрей спит, то посетитель будит его и садится в кресло, где его будут стричь.

Реализация: На языке C с использованием PTHREADS. Реализация почти полностью соответствует описанию решения задачи из книги *The Little Book of Semaphores, Allen B. Downey.*

Исходный код программы

Listing 1: Barbershop problem simulation program source

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <termios.h>
4  #include <unistd.h>
5
6  #include <pthread.h>
7  #include <semaphore.h>
8
9  int chair_number, haircut_duration;
10
11 int customers, barbershop;
12 pthread_mutex_t mutex_customers, mutex_barbershop;
13 sem_t customer, barber, haircut;
14
15 void printHelp( char *program_name )
16 {
17         printf("Barbershop problem simulation program\n"
18                         "Usage: %s <chair number> <haircut duration in seconds>\n", program_name)
                                ;
19 }
20
21 int mygetch( )
22 {
23         struct termios oldt, newt;
24         int ch;
25
26         tcgetattr( STDIN_FILENO, &oldt );
27         newt = oldt;
28         newt.c_lflag &= ~( ICANON | ECHO );
29         tcsetattr( STDIN_FILENO, TCSANOW, &newt );
30         ch = getchar();
31         tcsetattr( STDIN_FILENO, TCSANOW, &oldt );
32
33         return ch;
34 }
35
36 void get_haircut( int id )
37 {
38         printf("Barber is mastering Mr. %i 's haircut.\n", id);
39         sleep(haircut_duration);
40         sem_post(&haircut);
41 }
42
43 void cut_hair( void )
44 {
45         sem_wait(&haircut);
46         printf("Barber finished another haircut.\n");
47 }
48
49 void *customer_procedure( void *arg )
50 {
51         pthread_t self_thread_id = pthread_self();
```

```
52
53          unsigned int self_id = (unsigned long)self_thread_id % 30;
54
55          printf("Mr. %i arrives to the barbershop.\n", self_id);
56
57          pthread_mutex_lock(&mutex_customers);
58
59          if (customers == chair_number)
60          {
61                  printf("There is no free chair to wait for Mr. %i. He leaves the barbershop.\n",
                        self_id);
62                  pthread_mutex_unlock(&mutex_customers);
63                  pthread_exit(NULL);
64          }
65
66          customers++;
67          printf("Mr. %i took a seat. (%i/%i)\n", self_id, customers, chair_number);
68
69          pthread_mutex_unlock(&mutex_customers);
70
71          sem_post(&customer);
72          sem_wait(&barber);
73
74          pthread_mutex_lock(&mutex_customers);
75
76          customers--;
77
78          pthread_mutex_unlock(&mutex_customers);
79
80          printf("Mr. %i is going to get a new haircut. (%i/%i)\n", self_id, customers, chair_number
                );
81
82          get_haircut(self_id);
83
84          printf("Mr. %i got a new haircut. He leaves the barbershop. (%i/%i)\n", self_id, customers
                , chair_number);
85
86          pthread_exit(NULL);
87  }
88
89  void *barber_procedure( void *arg )
90  {
91          printf("Barbershop is open.\n");
92
93          while (1)
94          {
95                  int cur_customers;
96
97                  pthread_mutex_lock(&mutex_customers);
98                  cur_customers = customers;
99                  pthread_mutex_unlock(&mutex_customers);
100
101                 if (cur_customers == 0)
102                         printf("Barber is sleeping, waiting for the customer.\n");
103
104                 sem_wait(&customer);
105
106                 if (cur_customers == 0)
107                         printf("Barber woke up. Going to work.\n");
108
109                 sem_post(&barber);
110                 cut_hair();
111         }
112
113         return NULL;
114 }
115
116 void init_semaphores( void )
117 {
118         barbershop = 1;
119         customers = 0;
```

```
120         pthread_mutex_init(&mutex_barbershop, NULL);
121         pthread_mutex_init(&mutex_customers, NULL);
122         sem_init(&barber, 0, 0);
123         sem_init(&customer, 0, 0);
124         sem_init(&haircut, 0, 0);
125 }
126
127 void destroy_semaphores( void )
128 {
129         pthread_mutex_destroy(&mutex_barbershop);
130         pthread_mutex_destroy(&mutex_customers);
131         sem_destroy(&barber);
132         sem_destroy(&customer);
133         sem_destroy(&haircut);
134 }
135
136 void new_customer( void )
137 {
138         pthread_t customer_thread;
139
140         pthread_create(&customer_thread, NULL, customer_procedure, NULL);
141 }
142
143 int main( int argc, char *argv[] )
144 {
145         int flag = 1;
146
147         if (argc < 3)
148         {
149                 printHelp(argv[0]);
150                 return EXIT_FAILURE;
151         }
152
153         pthread_t barber_thread;
154
155         chair_number = atoi(argv[1]);
156         haircut_duration = atoi(argv[2]);
157
158         pthread_create(&barber_thread, NULL, barber_procedure, NULL);
159
160         init_semaphores();
161
162         while (flag)
163         {
164                 char c = mygetch();
165
166                 switch (c)
167                 {
168                 case 'c':
169                         flag = 0;
170                         pthread_cancel(barber_thread);
171                         break;
172                 case 'a':
173                         new_customer();
174                         break;
175                 }
176         }
177
178         destroy_semaphores();
179
180         pthread_exit(NULL);
181
182         return EXIT_SUCCESS;
183 }
```