

OPERATING SYSTEMS HOMEWORK #1

김현준 (2012003954), 한양대학교

2015-05-07

The Dining Philosophers Problem

Comments on this homework:

To be written

References:

1. Full HW description:

<https://github.com/yoloseem/os-homeworks/blob/master/hw1/README.md>

2. Raw source codes:

<https://github.com/yoloseem/os-homeworks/tree/master/hw1>

3. Commit history:

<https://github.com/yoloseem/os-homeworks/commits/master/hw1>

Screenshots:

To be added

Source codes:

Makefile

```
1  # Makefile
2  #
3  all: philo
4
5  clean:
6      ${RM} philo
7
8  philo:
9      ${CC} -o philo philo.c -Wno-int-to-void-pointer-cast
```

philo.c (Main source code)

```
1  /* philo.c */
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <limits.h>
5  #include <unistd.h>
6  #include <sys/time.h>
```

```

7  #include <pthread.h>
8  #include <semaphore.h>
9
10 #define DEBUG 0
11 #define debug_print(fmt, ...) \
12     do { if (DEBUG) fprintf(stderr, fmt, __VA_ARGS__); } while (0)
13
14 #define MAX(a, b) (a)>(b)?(a):(b)
15 #define MIN(a, b) (a)<(b)?(a):(b)
16
17 #define HUNGRY 0
18 #define EATING 1
19 #define THINKING 2
20
21 #define NUM_PHIL 5
22 #define EXEC_TIME 600
23
24 typedef struct philosopher {
25     unsigned short numEat;
26     int state;
27     long wait;
28 } philosopher;
29 philosopher phil[NUM_PHIL];
30 char *verboseStates[] = {"HUNGRY", "EATING", "THINKING"};
31
32 sem_t chopstick[NUM_PHIL];
33 sem_t lock;
34
35 int idlewait () // 10~500 msec wait
36 {
37     int sleepTimeMS = (rand() % 491 + 10);
38     usleep(sleepTimeMS * 1000);
39     return sleepTimeMS;
40 }
41
42 unsigned int tick () { // get current time (msec)
43     struct timeval tv;
44     gettimeofday(&tv, (void*)0);
45     return tv.tv_sec * (unsigned int)1000 + tv.tv_usec / 1000;
46 }
47
48 void initPhil (void) {
49     unsigned short i;
50     for (i=0; i<NUM_PHIL; i++) {
51         phil[i].numEat = 0;
52         phil[i].state = THINKING;
53         phil[i].wait = 0;
54         sem_init(&chopstick[i], 0, 1);
55     }
56 }
57

```

```

58 void* dining (void* arg) {
59     unsigned short i;
60     unsigned short left, right;
61     unsigned int start_time;
62     unsigned int start_hungry, end_hungry;
63     unsigned short phil_i = (int)arg;
64     philosopher* curphil = &phil[phil_i];
65     left = phil_i;
66     right = (phil_i + 1) % NUM_PHIL;
67
68     start_time = tick();
69     while ((tick() - start_time) / 1000 < EXEC_TIME) {
70         // initially/still THINKING
71         idlewait();
72
73         // HUNGRY
74         curphil->state = HUNGRY;
75         start_hungry = tick();
76         // HUNGRY -- To eat, acquires chopsticks
77         sem_wait(&lock);
78         sem_wait(&chopstick[left]);
79         sem_wait(&chopstick[right]);
80         end_hungry = tick();
81
82         // EATING
83         curphil->state = EATING;
84         curphil->wait += (end_hungry - start_hungry);
85         curphil->numEat++;
86         idlewait();
87         // EATING -- To think(and not hungry), release chopsticks
88         sem_post(&chopstick[left]);
89         sem_post(&chopstick[right]);
90         sem_post(&lock);
91
92         // Stop EATING and go THINKING
93         curphil->state = THINKING;
94     }
95
96     return (void*)NULL;
97 }
98
99 int main (void) {
100     pthread_t t[NUM_PHIL];
101     unsigned short i, args[NUM_PHIL], minCount = USHRT_MAX, maxCount = 0;
102     long start, end, minWait = LONG_MAX, maxWait = 0, waitAVG = 0, waitVar = 0;
103     double countAVG = 0, countVar = 0;
104     void *t_return = NULL;
105
106     srand(time(NULL));
107     start = tick();
108     initPhil();

```

```

109     sem_init(&lock, 0, NUM_PHIL - 1);
110
111     for (i=0; i<NUM_PHIL; i++) {
112         args[i] = i;
113         pthread_create(&t[i], NULL, dining, (void*)args[i]);
114     }
115     for (i=0; i<NUM_PHIL; i++) {
116         pthread_join(t[i], &t_return);
117     }
118     end = tick();
119
120     for (i=0; i<NUM_PHIL; i++)
121         sem_destroy(&chopstick[i]);
122     sem_destroy(&lock);
123
124     for (i=0; i<NUM_PHIL; i++) {
125         printf("Philosopher %d eating count : %d\n", i, phil[i].numEat);
126         printf("Philosopher %d waiting time in HUNGRY state : %ld.%03ld sec",
127             i, phil[i].wait / 1000, phil[i].wait % 1000);
128         printf("\n\n");
129         countAVG += phil[i].numEat;
130
131         minCount = MIN(minCount, phil[i].numEat);
132         maxCount = MAX(maxCount, phil[i].numEat);
133         waitAVG += phil[i].wait;
134         minWait = MIN(minWait, phil[i].wait);
135         maxWait = MAX(maxWait, phil[i].wait);
136     }
137     countAVG /= NUM_PHIL;
138     waitAVG /= NUM_PHIL;
139
140     for (i=0; i<NUM_PHIL; i++) {
141         countVar += (countAVG - phil[i].numEat) * (countAVG - phil[i].numEat);
142         waitVar += (waitAVG - phil[i].wait) * (waitAVG - phil[i].wait);
143     }
144     countVar /= (NUM_PHIL - 1);
145     waitVar /= (NUM_PHIL - 1);
146
147     printf("Min count : %d\n", minCount);
148     printf("Max count : %d\n", maxCount);
149     printf("AVG count : %.3f\n", countAVG);
150     printf("Count variance : %.3f\n\n", countVar);
151     printf("Min wait time in HUNGRY state : %ld.%03ld sec\n",
152         minWait / 1000, minWait % 1000);
153     printf("Max wait time in HUNGRY state : %ld.%03ld sec\n",
154         maxWait / 1000, maxWait % 1000);
155     printf("AVG wait time in HUNGRY state : %ld.%03ld sec\n",
156         waitAVG / 1000, waitAVG % 1000);
157     printf("Variance wait time in HUNGRY state : %ld.%06ld sec\n\n",
158         waitVar / 1000000, (waitVar % 1000000) / 1000);
159     printf("Total run time : %ld.%03ld sec\n\n",

```

```
160         (end - start)/ 1000, (end - start)% 1000);  
161     return 0;  
162 }
```