Operating Systems Homework #2

김현준 (2012003954), 한양대학교 2015-05-07

The Dining Philosophers Problem

Comments on this homework:

유명한 철학자 문제를 직접 구현해볼 수 있어 즐거운 과제였습니다. 처음에는 강의 시간에 교수님께서 예로 들어주신 것처럼 짝수번째 철학자와 홀수번째 철학자가 서로 다른 방향의 젓가락을 먼저 차지하는 전략으로 구현했고, 이후에는 최대 N - 1명의 철학자만 동시에 젓가락을 집으려고 시도할 수 있도록 제한하는 방법으로 다시 구현하였습니다(https://github.com/yoloseem/os-homeworks/commit/682dbaf643b1028d4823e4239e0d96e48c8c21d8). 이와 같이 여러 방법을 시도해보면서 문제를 해결하는데에는 다양한 방법이 존재할 수 있음을 다시금 경험할 수 있었고, 그와 동시에 여러 해결책을 생각해내기 위해서는 기반 지식과 경험 역시 탄탄해야하겠다고 생각했습니다. 예컨대, 최대 N - 1명을 허용하는데에 Semaphore를 사용하는 방법은, 단순히 단일 트랜잭션에 대한 Lock으로만, 즉 Binary Semaphore로서만 이해하고 있던 저로서는 이번 과제를 하면서 제대로 알 수 있게된 부분입니다.

과제 내용과 별개로 기억에 남는 것은, 과제 구현을 시작할 때부터 Makefile을 만들어두었더니 매 코드 변경시마다 컴파일 및 실행을 간편하게 할 수 있어서 좋았습니다.

References:

1. Full HW description:

   https://github.com/yoloseem/os-homeworks/blob/master/hw2/README.md

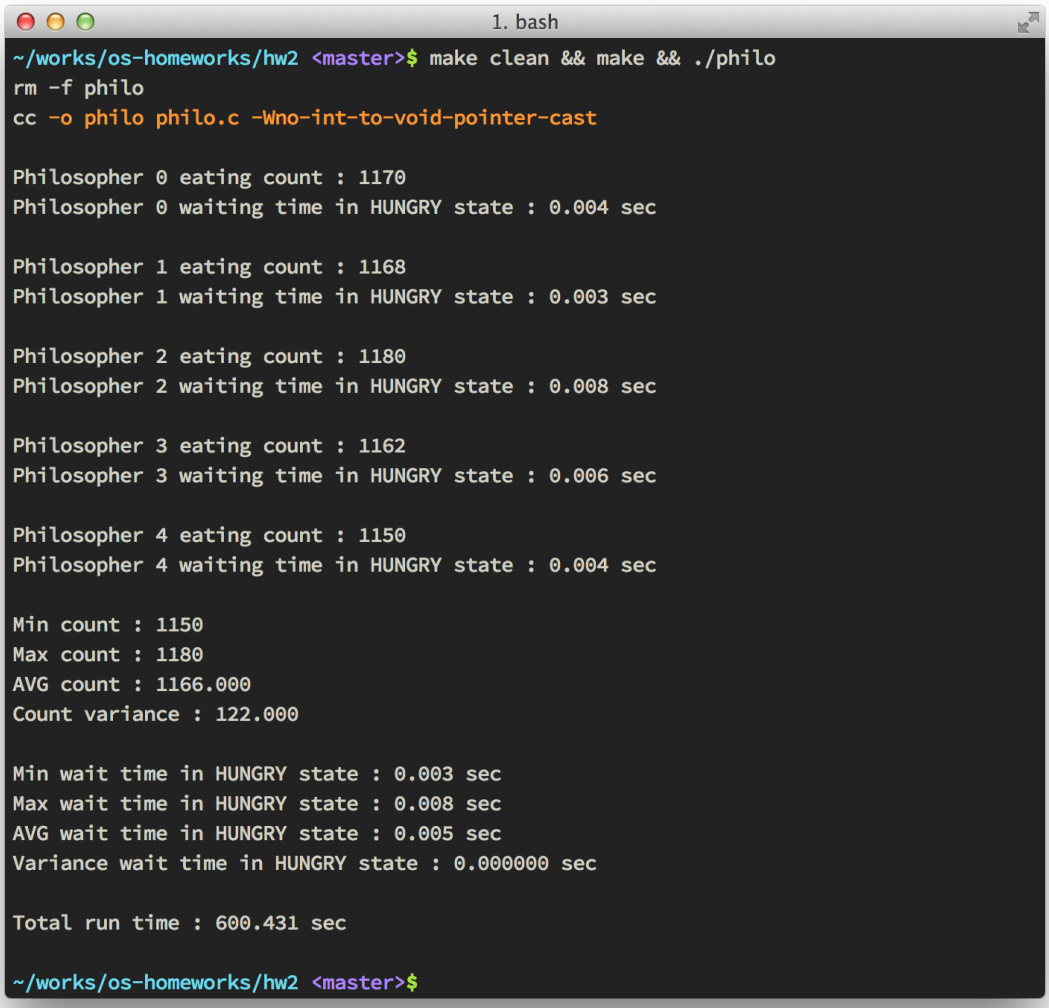
2. Raw source codes:

   https://github.com/yoloseem/os-homeworks/tree/master/hw2

3. Commit history:

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

Screenshot:



Source codes:

Makefile

```
1  # Makefile
2  #
3  all: philo
4
5  clean:
6          ${RM} philo
7
8  philo:
9          ${CC} -o philo philo.c -lpthread
```

philo.c (Main source code)

```
1  /* philo.c */
2  #include <stdio.h>
```

```c
#include <stdlib.h>
#include <limits.h>
#include <unistd.h>
#include <sys/time.h>
#include <pthread.h>
#include <semaphore.h>

#define MAX(a, b) (a)>(b)?(a):(b)
#define MIN(a, b) (a)<(b)?(a):(b)

#define HUNGRY 0
#define EATING 1
#define THINKING 2

#define NUM_PHIL 5
#define EXEC_TIME 600

typedef struct philosopher {
    unsigned short numEat;
    int state;
    long wait;
} philosopher;
philosopher phil[NUM_PHIL];
char *verboseStates[] = {"HUNGRY", "EATING", "THINKING"};

// chopstick semaphores: Binary semaphores for each chopsticks
sem_t chopstick[NUM_PHIL];
// counting semaphore: Keep up to n -1 philosophers trying to acquire chopstick
sem_t lock;

int idlewait ()  // 10~500 msec wait
{
    int sleepTimeMS = (rand() % 491 + 10);
    usleep(sleepTimeMS * 1000);
    return sleepTimeMS;
}

unsigned int tick () {  // get current time (msec)
    struct timeval tv;
    gettimeofday(&tv, (void*)0);
    return tv.tv_sec * (unsigned int)1000 + tv.tv_usec / 1000;
}

void initPhil (void) {
    // 1. Set up philosophers' initial states as specified in HW description
    // 2. Initialize chopstick semaphores
    unsigned short i;
    for (i=0; i<NUM_PHIL; i++) {
        phil[i].numEat = 0;
        phil[i].state = THINKING;
        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)(intptr_t)arg;
64      philosopher* curphil = &phil[phil_i]; // reference to current philosopher
65      left = phil_i;
66      right = (phil_i + 1) % NUM_PHIL;
67
68      start_time = tick();
69      // Repeat think-hungry-eating cycle during given execution time in secs
70      while ((tick() - start_time) / 1000 < EXEC_TIME) {
71          // initially or still in THINKING state
72          idlewait();
73
74          // Got into HUNGRY state
75          curphil->state = HUNGRY;
76          start_hungry = tick();
77          // To eat, acquires chopsticks
78          // 1. Wait for my turn
79          //    (up to n-1 philosophers are permitted to acquire chopsticks)
80          sem_wait(&lock);
81          // 2. Wait and acquire both chopsticks
82          sem_wait(&chopstick[left]);
83          sem_wait(&chopstick[right]);
84          // 3. Timing
85          end_hungry = tick();
86
87          // Got into EATING state
88          curphil->state = EATING;
89          curphil->wait += (end_hungry - start_hungry);
90          curphil->numEat++;
91          idlewait();
92          // To think(and not hungry), release chopsticks
93          sem_post(&chopstick[left]);
94          sem_post(&chopstick[right]);
95          sem_post(&lock);
96
97          // Stop EATING and go to THINKING state again
98          curphil->state = THINKING;
99      }
100
101     return (void*)NULL;
102 }
103
104 int main (void) {
```

```
105     pthread_t t[NUM_PHIL];
106     unsigned short i, args[NUM_PHIL], minCount = USHRT_MAX, maxCount =0;
107     long start, end, minWait = LONG_MAX, maxWait = 0, waitAVG = 0, waitVar = 0;
108     double countAVG = 0, countVar = 0;
109     void *t_return = NULL;
110
111     srand(time(NULL));
112     start = tick();
113     initPhil();
114     // Initialize philosopher-counting semaphore
115     sem_init(&lock, 0, NUM_PHIL - 1);
116
117     // Spawn philosopher threads
118     for (i=0; i<NUM_PHIL; i++) {
119         args[i] = i;
120         pthread_create(&t[i], NULL, dining, (void*)(intptr_t)args[i]);
121     }
122     for (i=0; i<NUM_PHIL; i++) {
123         pthread_join(t[i], &t_return);
124     }
125     end = tick(); // Timing
126
127     // Destory all used semaphores
128     for (i=0; i<NUM_PHIL; i++)
129         sem_destroy(&chopstick[i]);
130     sem_destroy(&lock);
131
132     for (i=0; i<NUM_PHIL; i++) {
133         printf("Philosopher %d eating count : %d\n", i, phil[i].numEat);
134         printf("Philosopher %d waiting time in HUNGRY state : %ld.%03ld sec",
135                i, phil[i].wait / 1000, phil[i].wait % 1000);
136         printf("\n\n");
137         countAVG += phil[i].numEat;
138
139         minCount = MIN(minCount, phil[i].numEat);
140         maxCount = MAX(maxCount, phil[i].numEat);
141         waitAVG += phil[i].wait;
142         minWait = MIN(minWait, phil[i].wait);
143         maxWait = MAX(maxWait, phil[i].wait);
144     }
145     countAVG /= NUM_PHIL;
146     waitAVG /= NUM_PHIL;
147
148     for (i=0; i<NUM_PHIL; i++) {
149         countVar += (countAVG - phil[i].numEat) * (countAVG - phil[i].numEat);
150         waitVar += (waitAVG - phil[i].wait) * (waitAVG - phil[i].wait);
151     }
152     countVar /= (NUM_PHIL - 1);
153     waitVar /= (NUM_PHIL - 1);
154
155     printf("Min count : %d\n", minCount);
```

```
156     printf("Max count : %d\n", maxCount);
157     printf("AVG count : %.3f\n", countAVG);
158     printf("Count variance : %.3f\n\n", countVar);
159     printf("Min wait time in HUNGRY state : %ld.%03ld sec\n",
160            minWait / 1000, minWait % 1000);
161     printf("Max wait time in HUNGRY state : %ld.%03ld sec\n",
162            maxWait / 1000, maxWait % 1000);
163     printf("AVG wait time in HUNGRY state : %ld.%03ld sec\n",
164            waitAVG / 1000, waitAVG % 1000);
165     printf("Variance wait time in HUNGRY state : %ld.%06ld sec\n\n",
166            waitVar / 1000000, (waitVar % 1000000) / 1000);
167     printf("Total run time : %ld.%03ld sec\n\n",
168           (end - start)/ 1000, (end - start)% 1000);
169     return 0;
170 }
```