Operating Systems Homework #2

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/hw2/README.md

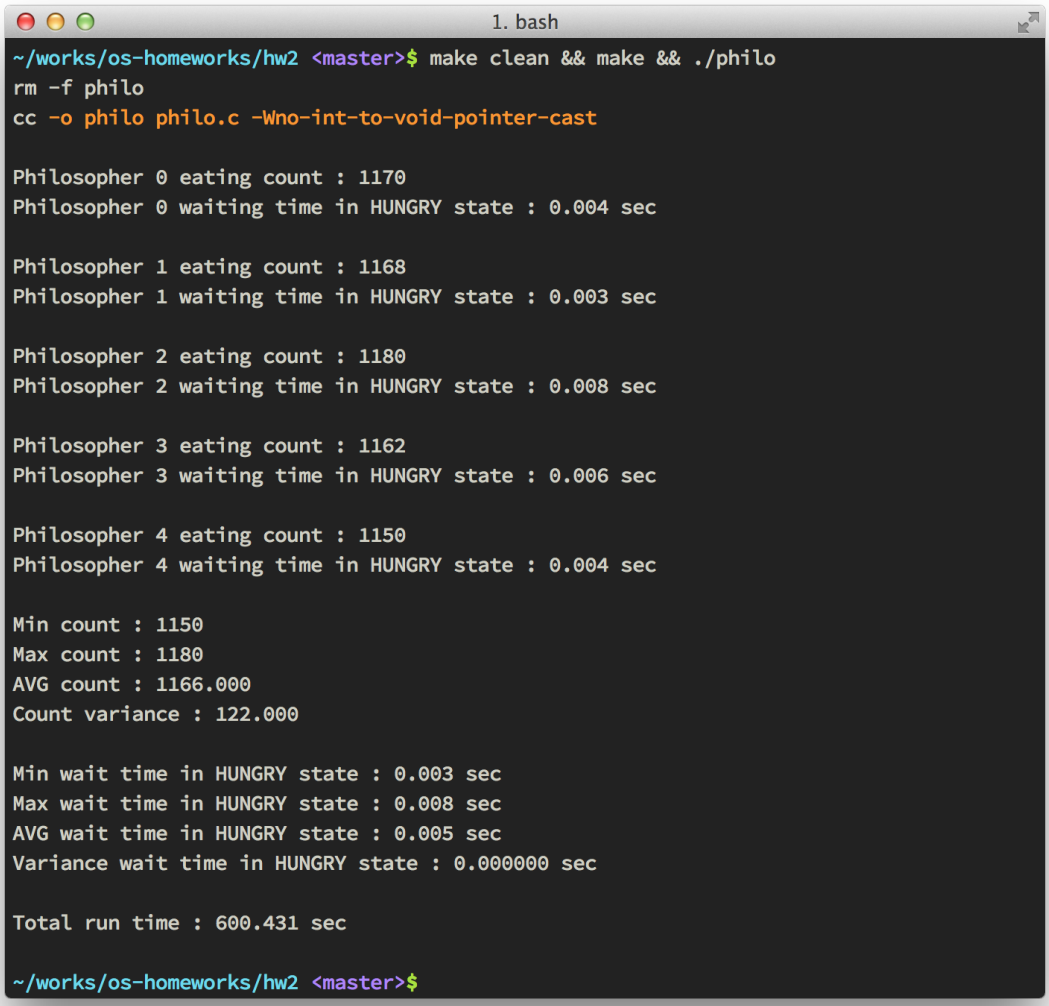
2. Raw source codes:

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

3. Commit history:

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

Screenshot:



```
~/works/os-homeworks/hw2 <master>$ make clean && make && ./philo
rm -f philo
cc -o philo philo.c -Wno-int-to-void-pointer-cast

Philosopher 0 eating count : 1170
Philosopher 0 waiting time in HUNGRY state : 0.004 sec

Philosopher 1 eating count : 1168
Philosopher 1 waiting time in HUNGRY state : 0.003 sec

Philosopher 2 eating count : 1180
Philosopher 2 waiting time in HUNGRY state : 0.008 sec

Philosopher 3 eating count : 1162
Philosopher 3 waiting time in HUNGRY state : 0.006 sec

Philosopher 4 eating count : 1150
Philosopher 4 waiting time in HUNGRY state : 0.004 sec

Min count : 1150
Max count : 1180
AVG count : 1166.000
Count variance : 122.000

Min wait time in HUNGRY state : 0.003 sec
Max wait time in HUNGRY state : 0.008 sec
AVG wait time in HUNGRY state : 0.005 sec
Variance wait time in HUNGRY state : 0.000000 sec

Total run time : 600.431 sec

~/works/os-homeworks/hw2 <master>$
```

Source codes:

Makefile

```makefile
# Makefile
#
all: philo

clean:
	${RM} philo

philo:
	${CC} -o philo philo.c -lpthread
```

philo.c (Main source code)

```c
/* philo.c */
#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"};

sem_t chopstick[NUM_PHIL];
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) {
    unsigned short i;
    for (i=0; i<NUM_PHIL; i++) {
        phil[i].numEat = 0;
        phil[i].state = THINKING;
        phil[i].wait = 0;
        sem_init(&chopstick[i], 0, 1);
    }
}

```

```c
void* dining (void* arg) {
    unsigned short i;
    unsigned short left, right;
    unsigned int start_time;
    unsigned int start_hungry, end_hungry;
    unsigned short phil_i = (int)(intptr_t)arg;
    philosopher* curphil = &phil[phil_i];
    left = phil_i;
    right = (phil_i + 1) % NUM_PHIL;

    start_time = tick();
    while ((tick() - start_time) / 1000 < EXEC_TIME) {
        // initially/still THINKING
        idlewait();

        // HUNGRY
        curphil->state = HUNGRY;
        start_hungry = tick();
        // HUNGRY -- To eat, acquires chopsticks
        sem_wait(&lock);
        sem_wait(&chopstick[left]);
        sem_wait(&chopstick[right]);
        end_hungry = tick();

        // EATING
        curphil->state = EATING;
        curphil->wait += (end_hungry - start_hungry);
        curphil->numEat++;
        idlewait();
        // EATING -- To think(and not hungry), release chopsticks
        sem_post(&chopstick[left]);
        sem_post(&chopstick[right]);
        sem_post(&lock);

        // Stop EATING and go THINKING
        curphil->state = THINKING;
    }

    return (void*)NULL;
}

int main (void) {
    pthread_t t[NUM_PHIL];
    unsigned short i, args[NUM_PHIL], minCount = USHRT_MAX, maxCount =0;
    long start, end, minWait = LONG_MAX, maxWait = 0, waitAVG = 0, waitVar = 0;
    double countAVG = 0, countVar = 0;
    void *t_return = NULL;

    srand(time(NULL));
    start = tick();
    initPhil();
```

```
    sem_init(&lock, 0, NUM_PHIL - 1);

    for (i=0; i<NUM_PHIL; i++) {
        args[i] = i;
        pthread_create(&t[i], NULL, dining, (void*)(intptr_t)args[i]);
    }
    for (i=0; i<NUM_PHIL; i++) {
        pthread_join(t[i], &t_return);
    }
    end = tick();

    for (i=0; i<NUM_PHIL; i++)
        sem_destroy(&chopstick[i]);
    sem_destroy(&lock);

    for (i=0; i<NUM_PHIL; i++) {
        printf("Philosopher %d eating count : %d\n", i, phil[i].numEat);
        printf("Philosopher %d waiting time in HUNGRY state : %ld.%03ld sec",
               i, phil[i].wait / 1000, phil[i].wait % 1000);
        printf("\n\n");
        countAVG += phil[i].numEat;

        minCount = MIN(minCount, phil[i].numEat);
        maxCount = MAX(maxCount, phil[i].numEat);
        waitAVG += phil[i].wait;
        minWait = MIN(minWait, phil[i].wait);
        maxWait = MAX(maxWait, phil[i].wait);
    }
    countAVG /= NUM_PHIL;
    waitAVG /= NUM_PHIL;

    for (i=0; i<NUM_PHIL; i++) {
        countVar += (countAVG - phil[i].numEat) * (countAVG - phil[i].numEat);
        waitVar += (waitAVG - phil[i].wait) * (waitAVG - phil[i].wait);
    }
    countVar /= (NUM_PHIL - 1);
    waitVar /= (NUM_PHIL - 1);

    printf("Min count : %d\n", minCount);
    printf("Max count : %d\n", maxCount);
    printf("AVG count : %.3f\n", countAVG);
    printf("Count variance : %.3f\n\n", countVar);
    printf("Min wait time in HUNGRY state : %ld.%03ld sec\n",
           minWait / 1000, minWait % 1000);
    printf("Max wait time in HUNGRY state : %ld.%03ld sec\n",
           maxWait / 1000, maxWait % 1000);
    printf("AVG wait time in HUNGRY state : %ld.%03ld sec\n",
           waitAVG / 1000, waitAVG % 1000);
    printf("Variance wait time in HUNGRY state : %ld.%06ld sec\n\n",
           waitVar / 1000000, (waitVar % 1000000) / 1000);
    printf("Total run time : %ld.%03ld sec\n\n",
```

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