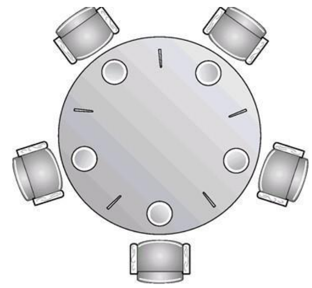
The Dining Philosophers Problem



 # The Dining Philosophers Problem

- Each philosopher repeats the following
  - sequence of states : {HUNGRY, EATING, THINKING}
- Init state : THINKING
  - 10~500 msec and then switches HUNGRY
- If he can eat, then immediately switches to EATING state
- He is in EATING state for 10~500 msec and then release both of the chopsticks
- After releasing both of the chopsticks, he switches to THINKING state

 # Result

```
Philosopher 0 eating count : 843
Philosopher 0 wait time in HUNGRY state (157.951 sec)

Philosopher 1 eating count : 842
Philosopher 1 wait time in HUNGRY state (171.384 sec)

Philosopher 2 eating count : 875
Philosopher 2 wait time in HUNGRY state (160.234 sec)

Philosopher 3 eating count : 849
Philosopher 3 wait time in HUNGRY state (167.027 sec)

Philosopher 4 eating count : 837
Philosopher 4 wait time in HUNGRY state (171.974 sec)

Min count 837
Max count 875
```

```
AVG count 849.200
Count Variance = 180.960

Min wait time (157.951 sec) in HUNGRY state
Max wait time (171.974 sec) in HUNGRY state
AVG wait time (165.714 sec) in HUNGRY state
Variance wait time (33.187007 sec) in HUNGRY state

Total Run time (600.042 sec)
```

- 제출 기한 : 2019.05.28 일 23:59
- 제출방법 : 보고서 파일 (한글,Word) 1, 프로그램 c 파일 1

  → zip 파일로 압축 제출

  - 압축파일 제목 (학번_이름_HW2.zip)
  - 보고서 양식 :
    - 문제정의
    - 실행 결과 화면,
    - 소스코드(각 함수 분석 내용 작성(주석) - 해결방법 설명)
  - 이메일 제출 : ssongsh@hanyang.ac.kr

Template Code

```
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>
#include <time.h>
#include <pthread.h>
#include <semaphore.h>
#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];
sem_t chopstick[NUM_PHIL];
// 10~500 msec wait
int idlewait()
{
    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)
{

/* ..................................................................
............................ */
}
void* dining(void* arg)
```

```
{
    unsigned short i;
    unsigned short left, right;
     unsigned int start_time;
    unsigned int start_hungry, end_hungry;

/* ..................................................................
............................ */

}

void 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;

    srand(time((void*)0));
    start = tick();
    initPhil();

    for(i=0;i<NUM_PHIL;i++)
        args[i]=i;


/* ..................................................................
............................ */

    end = tick();

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

```
        if(minWait > phil[i].wait)
            minWait = phil[i].wait;
        if(maxWait < phil[i].wait)
            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;
    waitVar /= NUM_PHIL;
    printf("Min count : %d\nMax count : %d\nAVG count : %.3f\nCount
variance : %.3f\n\n",
minCount, maxCount, countAVG, countVar);
    printf("Min wait time in HUNGRY state : %ld.%ld sec\nMax wait
time in HUNGRY state
: %ld.%ld sec\nAVG wait time in HUNGRY state : %ld.%ld sec\nVariance
wait time in
HUNGRY state : %ld.%ld sec\n\n",
           minWait / 1000, minWait % 1000, maxWait / 1000, maxWait %
1000, waitAVG /
1000,waitAVG % 1000, waitVar / 1000000, (waitVar % 1000000) / 1000);
    printf("Total run time : %ld.%ld sec\n\n", (end - start)/ 1000,
(end - start)% 1000);
}
```