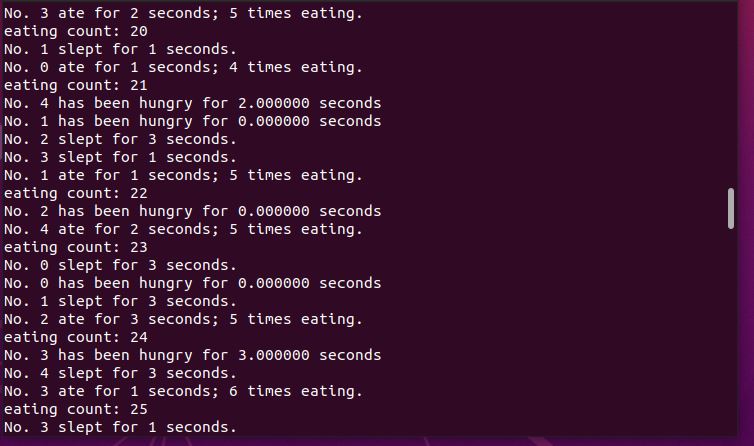
Group member：pickup(), putdown(), test(): 辛愷庭，main function: 江宥潔，run(): 易頡，refactoring, research: 戴陽

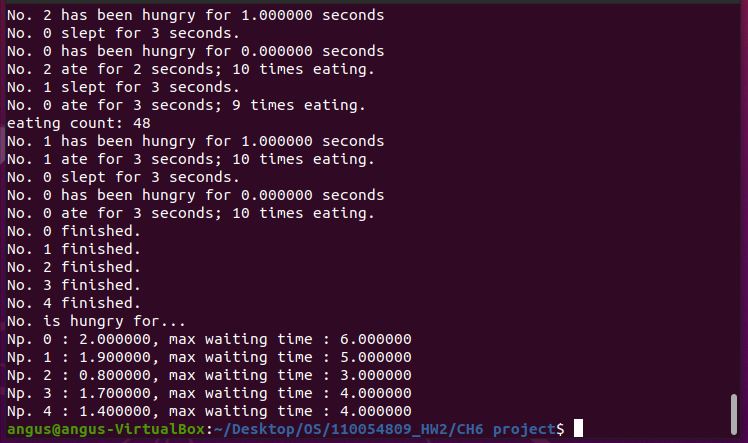
Environment：Linux

Instructions：

1. gcc eating\_philo.c -pthread -o eating\_philo.o

2. run ./ eating\_philo.o





Source code：

//pickup(), putdown(), test(): 辛愷庭

// main function: 江宥潔

// run(): 易頡

// refactoring, research: 戴陽

#include <stdio.h>

#include <pthread.h>

#include <time.h>

#include <stdlib.h>

#include <unistd.h>

int MAX\_EAT = 200;

int num = 5;

double avg\_waiting\_time[5];

double max\_waiting\_time[5];

enum

{

thinking,

hungry,

eating

} state[5];

pthread\_mutex\_t lock;

pthread\_cond\_t self[5];

pthread\_t philo[5];

int identity[5] = {0, 1, 2, 3, 4}; // anything like identity = [i for i in range(num)]??

int e\_count = 0;

void \*run\_phi(void \*arg);

void \*pickup(int);

void \*putdown(int);

void \*test(int);

int main()

{

pthread\_mutex\_init(&lock, NULL);

for (int i = 0; i < num; ++i)

{

state[i] = thinking;

pthread\_cond\_init(&self[i], NULL);

avg\_waiting\_time[i] = 0;

}

for (int i = 0; i < num; ++i)

{

pthread\_create(&philo[i], NULL, run\_phi, &identity[i]);

}

for (int i = 0; i < num; ++i)

{

pthread\_join(philo[i], NULL);

printf("No. %d finished.\n", i);

}

printf("No. is hungry for...\n");

for (int i = 0; i < num; i++)

{

printf("Np. %d : %f, max waiting time : %f\n", i, avg\_waiting\_time[i], max\_waiting\_time[i]);

}

return 0;

}

void \*run\_phi(void \*arg)

{

int \*tmp = (int \*)arg;

int no = \*tmp;

double total\_waiting\_time = 0, single\_waiting\_time = 0;

time\_t t\_start, t\_end, t;

printf("Philosopher %d is now working\n", no);

srand((unsigned)time(&t));

for (int i = 0; i < MAX\_EAT; ++i)

{

printf("eating count: %d\n", e\_count);

int some\_time = (rand()) % 3 + 1;

sleep(some\_time);

printf("No. %d slept for %d seconds.\n", no, some\_time);

time(&t\_start);

pickup(no);

time(&t\_end);

single\_waiting\_time = difftime(t\_end, t\_start);

if (single\_waiting\_time > max\_waiting\_time[no]) { //update max waiting time

max\_waiting\_time[no] = single\_waiting\_time;

}

printf("No. %d has been hungry for %f seconds\n", no, single\_waiting\_time);

sleep(some\_time);

printf("No. %d ate for %d seconds; %d times eating.\n", no, some\_time, i + 1);

putdown(no);

++e\_count;

total\_waiting\_time += single\_waiting\_time;

}

avg\_waiting\_time[no] = total\_waiting\_time / (double)MAX\_EAT;

return NULL;

}

// void \*eat(int no){printf("eating...")};

void \*pickup(int no)

{

pthread\_mutex\_lock(&lock);

state[no] = hungry;

pthread\_mutex\_unlock(&lock);

test(no); // merge 2 critical sections together?

pthread\_mutex\_lock(&lock);

if (state[no] != eating)

pthread\_cond\_wait(&self[no], &lock);

pthread\_mutex\_unlock(&lock);

return NULL;

}

void \*putdown(int no)

{

pthread\_mutex\_lock(&lock);

state[no] = thinking;

pthread\_mutex\_unlock(&lock);

test((no + 4) % 5);

test((no + 1) % 5);

}

void \*test(int no)

{

int rhs = (no + 1) % 5, lhs = (no + 4) % 5;

if (no % 2)

{

pthread\_mutex\_lock(&lock);

if (state[rhs] != eating && state[lhs] != eating && state[no] == hungry)

{

state[no] = eating;

pthread\_cond\_signal(&self[no]);

}

pthread\_mutex\_unlock(&lock);

}

else

{

pthread\_mutex\_lock(&lock);

if (state[rhs] != eating && state[lhs] != eating && state[no] == hungry)

{

state[no] = eating;

pthread\_cond\_signal(&self[no]);

}

pthread\_mutex\_unlock(&lock);

}

}