**گزارش تمرین امتیازی مربوط به آزمایش ششم آزمایشگاه سیستم­های عامل**

نگار موقتیان، 9831062

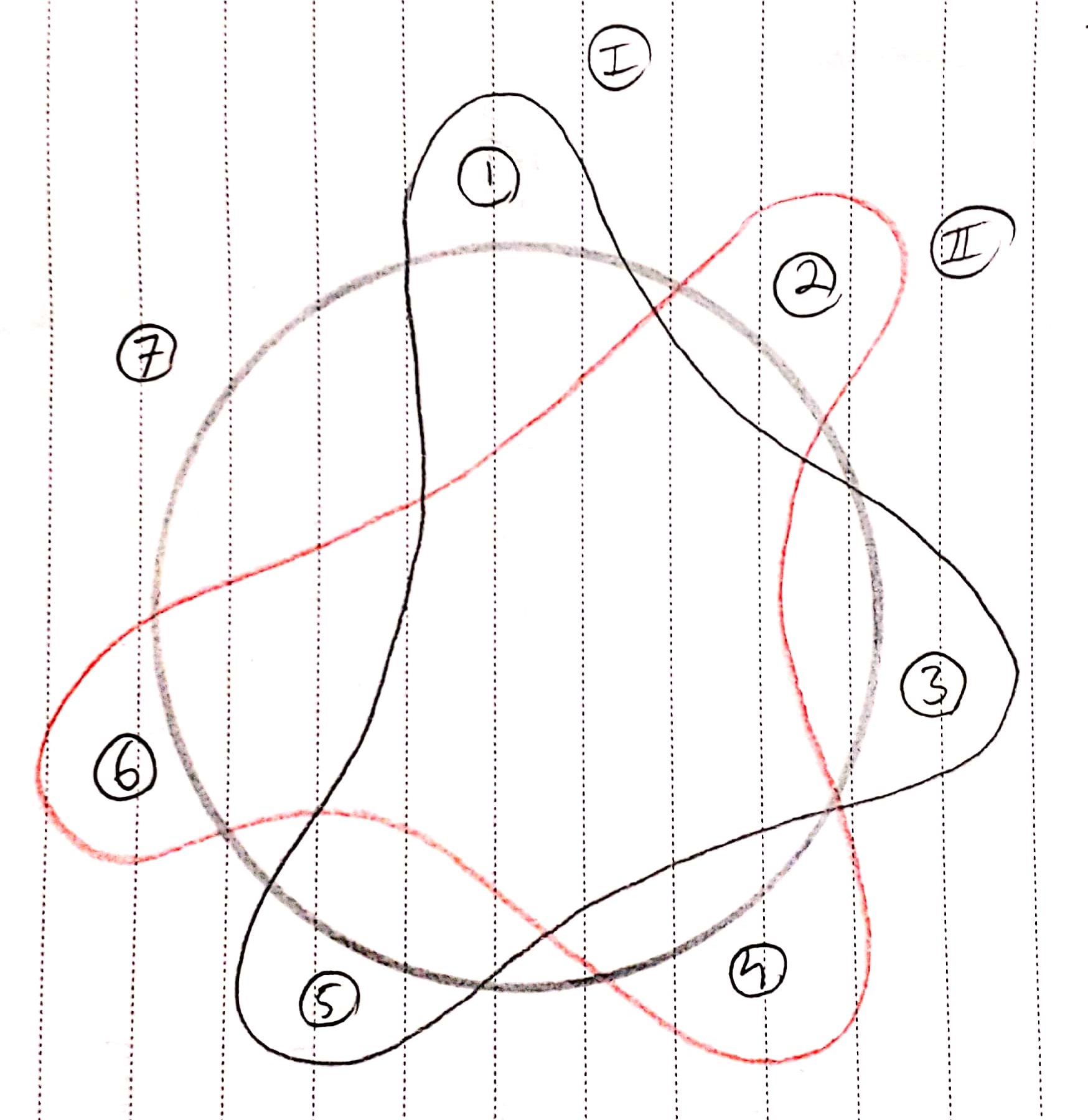
1. **ارائۀ الگوریتمی برای زمانبندی غذاخوردن فیلسوف­ها**

در این قسمت از آزمایش می­خواهیم برنامه­ای ارائه دهیم که فیلسوف­ها (که تعداد آن­ها عددی فرد است) بتوانند مطابق آن تصمیم بگیرند در هر لحظه باید فکر کنند و یا غذا بخورند.

در ابتدای برنامه کتابخانه­های مورد نیاز اضافه شده­اند.

سپس متغیرهای برنامه تعریف شده­اند. متغیر n ورودی­ای است که در ابتدای برنامه از کاربر دریافت می­شود. تعداد فیلسوف­ها در ادامه برابر خواهد بود با N = 2n + 1. سپس متغیر p با مقدار اولیۀ صفر تعریف شده که در هر نوبت اولین فیلسوفی که انتخاب می­کنیم را نشان می­دهد. این متغیر از اولین فیلسوف شروع کرده و در هر نوبت به صورت چرخشی یک عدد به جلو حرکت می­کند.

در تابع main برنامه ابتدا ورودی گرفته شده و سپس در یک حلقۀ بی­نهایت هر بار آرایۀ s در یک حلقۀ N تایی پر می­شود. به ازای هر خانۀ i ام از آرایه اگر مقدار خانه صفر بود فیلسوف باید فکر کند و اگر یک بود می­تواند غذا بخورد. به این ترتیب تصمیم گرفته می­شود که در هر نوبت کدام یک از فیلسوف­ها می­توانند با هم غذا بخورند (میزهای مجازی مشخص می­شوند). شیوۀ انتخاب فیلسوف­ها به طور کلی مانند زیر است. در این شکل دو نوبت از 7 نوبت نشان داده شده­است که در آن p برابر با 0 و 1 قرار می­گیرد. این فریم (که به رنگ مشکی و قرمز نمایش داده شده­است و همان میزهای مجازی ما هستند) به ترتیب همراه با متغیر p می­چرخد.



#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <time.h>

#include <string.h>

int n, N, p, counter;

int main() {

// get n as input. there will be N = 2n + 1 philosophers at the table

printf("> Please enter the value of 'n' (there will be 2n + 1 philosophers at the table): ");

scanf("%d", &n);

N = 2 \* n + 1; // number of philosophers

int s[N]; // for each philosopher: 0 -> think, 1 -> eat

while (1) { // decide which philasophors can eat at the same time in each round

// fill the s array, which indicates which philosophers should be eating and which philosophers should be thinking

memset(s, 0, sizeof s);

int curr = p;

if (N == 1)

s[0] = 1;

else

while (curr != ((p - 1) + N) % N)

s[curr] = 1, curr = (curr + 2) % N;

p = (p + 1) % N;

// print the virtual table's info

printf("Virtual table %d members: ", ++counter);

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

if (s[i])

printf("%d | ", i + 1);

printf("\b\b \n----------------------------------------------------\n");

// it takes 1 second for philosophers at the virtual table to finish eating

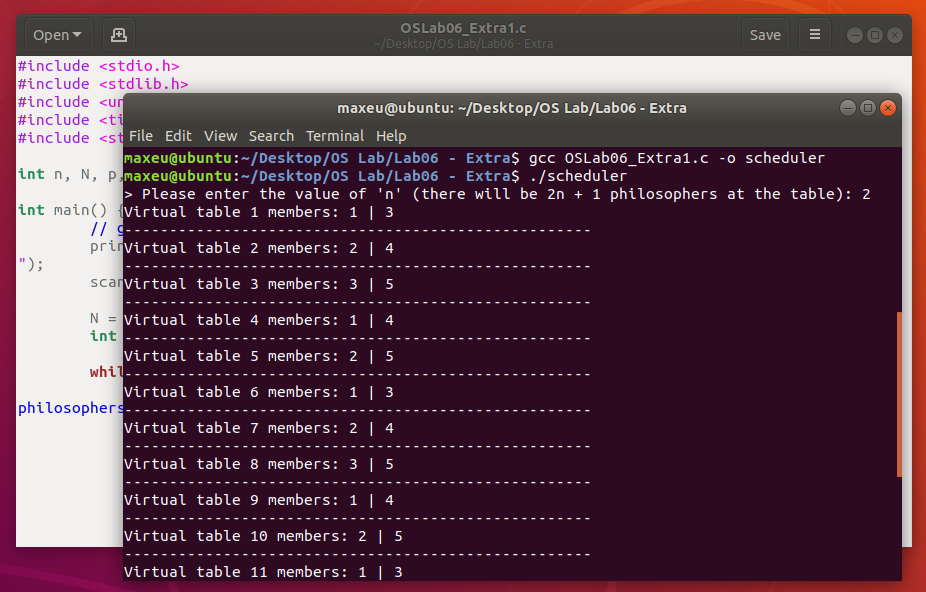
sleep(1);

}

return 0;

}

حال می­توانیم برنامۀ فوق را اجرا کنیم. خروجی برنامه به ازای 5 فیلسوف مانند زیر است.



انتخاب این گروه­ها منحصر به فرد نیست، به علاوه به این شیوه هر N نوبت یکبار فیلسوف­هایی که باید همزمان مشغول غذاخوردن باشند تکرار می­شوند.

در این روش به دلیل شیوۀ انتخاب میزهای مجازی می­توانیم مطمئن باشیم دیگر deadlock رخ نخواهد داد زیرا هرگز دو فیلسوف که کنار یکدیگر نشسته­اند همزمان برای غذا خوردن اقدام نمی­کنند، لذا فیلسوفی که نوبت آن است می­تواند بدون نگرانی دو چنگال دو طرف خود را بردارد.

1. **استفادۀ thread ها از خروجی قسمت قبل**

حال می­خواهیم از نتیجۀ برنامه ریزی قسمت قبل برای تعیین زمانی که thread فیلسوفان غذا می­خورند و یا فکر می­کنند استفاده کنیم.

تابع main این برنامه ترکیبی از کد مربوط به قسمت اصلی آزمایش ششم و کد قسمت قبل است. از کد قسمت قبل برای پر کردن آرایۀ s استفاده می­کنیم و از آن در تابع thinkAndEat استفاده می­کنیم (در واقع تابع main مانند یک زمانبند برای اجرای thread ها عمل می­کند). در این تابع هر فیلسوف منتظر می­شود تا خانۀ مربوط به آن از آرایۀ s برابر با یک شود، یعنی نوبت غذاخوردن آن شود. سپس قفل دو چنگال دو طرفش را بدست می­گیرد و تا زمانی که خانۀ مربوط به آن از آرایۀ s برابر با یک است و اجازۀ غذاخوردن دارد، غذا می­خورد و سپس قفل مربوط به چنگال­هایش را آزاد می­کند.

#include <time.h>

#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

#include <unistd.h>

#include <string.h>

#include <pthread.h>

# define MAX 100

pthread\_t philosophers[MAX];

pthread\_mutex\_t chopstick[MAX];

int s[MAX]; // for each philosopher: 0 -> think, 1 -> eat

int n, N, p;

void \*thinkAndEat(int n) {

while (1) {

printf("philosopher %d is thinking!!\n", n + 1);

while (!s[n]);

// acquire the chopsticks

pthread\_mutex\_lock(&chopstick[n]);

pthread\_mutex\_lock(&chopstick[(n + 1) % N]);

printf("philosopher %d is eating using chopstick[%d] and chopstick[%d]!!\n", n + 1, n, (n + 1) % N);

while (s[n]);

printf("philosopher %d finished eating!!\n", n + 1);

// release the chopsticks

pthread\_mutex\_unlock(&chopstick[n]);

pthread\_mutex\_unlock(&chopstick[(n + 1) % N]);

}

}

int main() {

// get n as input. there will be N = 2n + 1 philosophers at the table

printf("> Please enter the value of 'n' (there will be 2n + 1 philosophers at the table): ");

scanf("%d", &n);

N = 2 \* n + 1; // number of philosophers

// initiate the mutex locks used for accessing the chopsticks

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

pthread\_mutex\_init(&chopstick[i], NULL);

// create the threads representing the philosophers

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

pthread\_create(&philosophers[i], NULL, (void \*) thinkAndEat, (void \*)(intptr\_t) i);

while (1) { // decide which philasophors can eat at the same time in each round

// fill the s array, which indicates which philosophers should be eating and which philosophers should be thinking

memset(s, 0, sizeof s);

int curr = p;

if (N == 1)

s[0] = 1;

else

while (curr != ((p - 1) + N) % N)

s[curr] = 1, curr = (curr + 2) % N;

p = (p + 1) % N;

// we give 1 second to the philosophers at the virtual table to finish eating

sleep(1);

}

return 0;

}

با این روش دیگر مشکل deadlock نخواهیم داشت زیرا طبق زمانبندی مشخص شده دو فیلسوف کنار هم همزمان اقدام به غذا خوردن نخواهند کرد. البته همچنان نیاز به قفل کردن چنگال­ها تا اتمام غذا خوردن داریم زیرا ترتیب اجرای thread ها غیر قابل پیش­بینی است و ممکن است در main آرایۀ s آپدیت شده باشد اما thread قبلی همچنان از آن با خبر نشده و مشغول غذا خوردن است، پس باید تا متوجه شدن آن و آزادسازی منابع منتظر بمانیم.

خروجی این برنامه به ازای 5 فیلسوف مانند زیر خواهد بود. همانطور که مشاهده می­شود غذا خوردن فیلسوف­ها دقیقا مطابق با جدول زمانبندی قسمت قبل به ازای 5 فیلسوف انجام شده­است.

ابتدا فیلسوف­های 1 و 3، سپس 2 و 4، سپس 3 و 5 و ... غذا خورده­اند.

