## Question 1:

The c programming language implement for the Dining Philosophers program shown as follow:

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <pthread.h>

#include <semaphore.h>

#define PHILOSOPHER\_NUM 5

#define MAX\_SLEEP\_TIME 20000

sem\_t chopsticks[PHILOSOPHER\_NUM];

sem\_t max\_current\_sem;

int philosophers[PHILOSOPHER\_NUM] = {0,1,2,3,4};

void random\_time() {

  int cnt = rand() % MAX\_SLEEP\_TIME;

    for (int i = 0; i < cnt; i++) {

        int mm = 0;

        for (int j = 0; j < MAX\_SLEEP\_TIME; j++) {

            mm++;

        }

    }

}

void philosopher\_thinking(int i) {

    random\_time();

    printf("Philosopher: %d thinking time over.\n", i);

}

void philosopher\_eating(int i) {

    random\_time();

    printf("Philosopher: %d eating time over.\n", i);

}

void\* philosopher\_process(void\* data) {

    int index = \*((int\*)data);

    int leftHand = index, rightHand = (index + 1) % PHILOSOPHER\_NUM;

    while (1) {

        printf("Philosopher: %d is thinking a complex problem.\n", index);

        philosopher\_thinking(index);

        printf("Philosopher: %d is hungry now and want to eat.\n", index);

        sem\_wait(&max\_current\_sem);

        sem\_wait(&chopsticks[leftHand]);

        printf("Philosopher: %d got the %d chopsticks in hand, but can not to eat.\n", index, leftHand);

        sem\_wait(&chopsticks[rightHand]);

        printf("Philosopher: %d got the %d chopsticks in hand, ready to eat.\n", index, rightHand);

        philosopher\_eating(index);

        sem\_post(&chopsticks[leftHand]);

        printf("Philosopher: %d released the %d left chopsticks\n", index, leftHand);

        sem\_post(&max\_current\_sem);

        sem\_post(&chopsticks[rightHand]);

        printf("Philosopher: %d released the %d right chopsticks\n", index, rightHand);

    }

}

int main() {

    srand(time(NULL));

    pthread\_t philosopher\_threads[PHILOSOPHER\_NUM];

    for (int i = 0; i < PHILOSOPHER\_NUM; i++) {

        sem\_init(&chopsticks[i], 0, 1);

    }

    sem\_init(&max\_current\_sem, 0, 4);

    for (int i = 0; i < PHILOSOPHER\_NUM; i++) {

        pthread\_create(&philosopher\_threads[i], NULL, philosopher\_process, &philosophers[i]);

    }

    for (int i = 0; i < PHILOSOPHER\_NUM; i++) {

        pthread\_join(philosopher\_threads[i], NULL);

    }

    for (int i = 0; i < PHILOSOPHER\_NUM; i++) {

        sem\_destroy(&chopsticks[i]);

    }

    sem\_destroy(&max\_current\_sem);

    return 0;

}

## Question 2:

In the question 1, for the Dining Philosophers program, there would be four philosophers allowed to pick up the left chopsticks at the same time. Which shows that there would be at least one philosopher could eat the dinner (there is always one chopsticks left for the philosophers who was hungry and want to eat). After the philosophers who done the dinner and release the chopsticks, other philosophers who waiting to eat or waiting for another one chopstick could pick up the chopsticks and eat dinner. Such resource limitation could ensure at least one philosopher free for dinner and there would be no deadlock occurred.

## Question 3:

file-processes.cpp which modified for simulating shell command:

tr a-z A-Z < /etc/passwd

shown as follow:

#include <unistd.h>

#include <stdio.h>

#include <iostream>

#include <fcntl.h>

#include <sys/wait.h>

#include <sys/stat.h>

using namespace std;

int main(){

  pid\_t returnedValue = fork();

  if(returnedValue < 0){

    perror("error forking");

    return -1;

  }

  else if (returnedValue == 0){

    if(close(STDIN\_FILENO) < 0){

      perror("error closing standard output");

      return -1;

    }

    if(open("/etc/passwd", O\_RDONLY) < 0){

      perror("error opening my-processes");

      return -1;

    }

    execlp("tr", "tr", "a-z", "A-Z",  NULL);

    perror("error executing ps");

    return -1;

  }

  else {

    if(waitpid(returnedValue, 0, 0) < 0){

      perror("error waiting for child");

      return -1;

    }

    //cout << "Note the parent still has the old standard output." << endl;

  }

}

## Question 4:

The program which uses mmap to map the file which stored in the disk to the virtual memory and search for specified character ‘X’ would be shown as follow:

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/mman.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <time.h>

#define target 'X'

int main(int argc, char\* argv[]) {

    int fd;

    struct stat st;

    char\* mmapp;

    if (argc < 4) {

        printf("Usage: ./mmapp file1.txt file2.txt file3.txt.\n");

        return -1;

    }

    for (int i = 1; i < argc; i++) {

        printf("\n");

        clock\_t start, end;

        if ((fd = open(argv[i], O\_RDONLY)) < 0) {

            printf("Can not open the %s filr.\n", argv[i]);

            continue;

        }

        if ((fstat(fd, &st)) == -1) {

            printf("Can not get the structure of the file: %s.\n", argv[i]);

            return -1;

        }

        if ((mmapp = mmap(NULL, st.st\_size, PROT\_READ, MAP\_SHARED, fd, 0)) == (void\*)-1) {

            printf("mmap calling error for file: %s.\n", argv[i]);

            return -1;

        }

        start = clock();

        for (int j = 0; j < st.st\_size; j++) {

            if (mmapp[j] == target) {

                printf("Found the target character %c from the mmap file %s.\n", target, argv[i]);

                break;

            }

        }

        end = clock();

        double seconds = (double)(end - start) / CLOCKS\_PER\_SEC;

        printf("Running time for file %s is %.10fs.\n\n", argv[i], seconds);

        if ((munmap((void\*)mmapp, st.st\_size)) == -1) {

            printf("munmap error for file %s.\n", argv[i]);

            return -1;

        }

    }

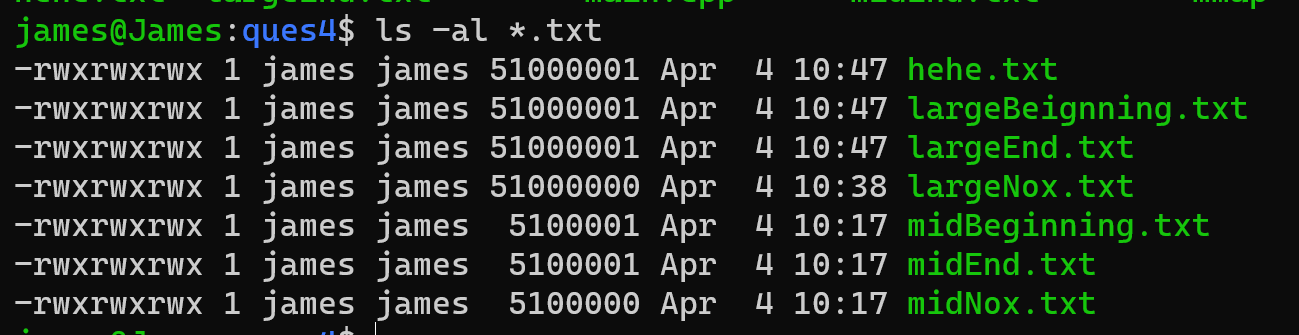
    return 0;

}

In the above c program language, filename would be passed to the program as the command line parameters, and then parsed by the program and opened to mmap to the virtual memory, and then the virtual memory could be thought as a continuous string with file size characters.

For timing the program on files of varying size, with ‘X’ in the beginning, end or no exist in the file, there would be two size files prepared for each condition:

1. File with at least 4MB size for ‘X’ in the beginning, end or no exist in the file
2. File with at least 40MB size for ‘X’ in the beginning, end or no exist in the file



mmap can be thought as a high efficiency method for file operation, which would cost less or no page loading from disk to main memory (such can be thought as the big cost to access the content in the file), and here we only record the searching time from the virtual memory for specified character without considering the map and unmap cost of file with the virtual memory.

And the running results shown as follow:

|  |  |  |
| --- | --- | --- |
|  | Middle size file (time/s) | Large size file (time/s) |
| Beginning with ‘X’ in the file | 0 | 0 |
| End with ‘X’ in the file | 0.015625 | 0.09375 |
| No ‘X’ in the file | 0.015625 | 0.109375 |

We can see that with the increase of size of the file, time used for searching the specified character would be increasing. Also for the character in the end of the file and no character in the file, the time nearly same with each other.

## Question 5:

The java class: BoundedBuffer from the chapter 4 would be appropriate to use for the holding area.

Condition Variables are used in the BoundedBuffer, which can be thought as a synchronization mechanism with mutex used in the style of monitors. TopicServer would occupy the holding area with using mutex for send new messages to the subscribers, and all these done, mutex is released and notify all waiting thread which would retrieve message from the holding area and sending these messages to the subscribers.