Group Project 2: Synchronization

CECS 326 – Operating Systems

You should submit the required deliverable materials on BeachBoard by **11:55pm, March 09th (Sunday), 2025.**

1. Description

Recall the dining-philosophers problem using monitors in our slides, this project involves implementing a solution to this problem using either POSIX mutex locks and condition variables in C/C++ or Java condition variables.

Both implementations will require creating five philosophers, each identified by a number 0 . . . 4. Each philosopher will run as a separate thread. Philosophers alternate between thinking and eating. To simulate both activities, have each thread sleep for a random period between one and three seconds.

I. POSIX

Thread creation using Pthreads (you can find how to use Pthreads with the APIs). When a philosopher wishes to eat, she invokes the function

pickup_forks (int philosopher_number)

where *philosopher_number* identifies the number of the philosopher wishing to eat. When a philosopher finishes eating, she invokes

return_forks (int philosopher_number)

Your implementation will require the use of POSIX condition variables, condition variables in Pthreads use the *pthread_cond_t* data type and are initialized using the *pthread_cond_init()* function. The following code creates and initializes a condition variable as well as its associated mutex lock:

```
pthread_mutex_t mutex;

pthread_cond_t cond_var;

pthread_mutex_init(&mutex,NULL);

pthread_cond_init(&cond_var,NULL);
```

The pthread_cond_wait() function is used for waiting on a condition variable. The following code illustrates how a thread can wait for the condition a == b to become true using a Pthread condition variable:

```
pthread_mutex_lock(&mutex);

while (a != b)

    pthread_cond_wait(&cond_var, &mutex);

pthread_mutex_unlock(&mutex);
```

The mutex lock associated with the condition variable must be locked before the pthread_cond_wait() function is called, since it is used to protect the data in the conditional clause from a possible race condition. Once this lock is acquired, the thread can check the condition. If the condition is not true, the thread then invokes pthread_cond_wait(), passing the mutex lock and the condition variable as parameters. Calling pthread_cond_wait() releases the mutex lock, thereby allowing another thread to access the shared data and possibly update its value so that the condition clause evaluates to true.

II. Java

When a philosopher wishes to eat, she invokes the method *takeForks(philosopherNumber)*, where *philosopherNumber* identifies the number of the philosopher wishing to eat. When a philosopher finishes eating, she invokes *returnForks(philosopherNumber)*.

Your solution will implement the following interface:

```
public interface DiningServer

{

  /* Called by a philosopher when it wishes to eat */

  public void takeForks(int philosopherNumber);

  /* Called by a philosopher when it is finished eating */

  public void returnForks(int philosopherNumber);

}
```

It will require the use of Java condition variables,

2. The Required Deliverable Materials

(1) A README file, which describes how we can compile and run your code.
(2) Your source code, should submit in the required format.
(3) Your short report, which discusses the design of your program.
(4) A recorded video shows the output and runtime

3. Submission Requirements

You need to strictly follow the instructions listed below:

1) This is a **group project**, please submit a .zip/.rar file that contains all files, only one submission from one group.

2) Make a **video** to record your code execution and outputs. The video should present your name or time as identification (You are suggested to upload the video to YouTube and put the link into your report).

3) The submission should include your **source code** and **project report**. Do not submit your binary code. Project report should contain your groupmates name and ID.

4) Your code must **be able to compile**; otherwise, you will receive a grade of zero.

5) Your code should not produce anything else other than the required information in the output file.

7) If you code is **partially completed**, please explain the details in the report what has been completed and the status of the missing parts, we will grade it based on the entire performance.

8) Provide **sufficient comments** in your code to help the TA understand your code. This is important for you to get at least partial credit in case your submitted code does not work properly.

Grading criteria:

Details	Points
Have a README file shows how to compile and test your submission	5 pts
Submitted code has proper comments to show the design	15 pts
Screen a ***video*** to record code execution and outputs	10 pts
Have a **report** (pdf or word) file explains the details of your entire design	20 pts
Report contains clearly individual contributions of your group mates	5 pts
Code can be compiled and shows correct outputs	45 pts

4. Policies

1) Late submissions will be graded based on our policy discussed in the course syllabus.
2) Code-level discussion is **prohibited**. We will use anti-plagiarism tools to detect violations of this policy.