CSC 33200 (L) - Operating Systems – Spring 2023

Lab 6: Process Synchronization
Date: 04/14/2023

Task1 **DUE: 04/27/2023**

The given **bank.c** program has 3 processes namely, the dad process and two son processes where a son withdrawing money from the bank and the dad depositing money in the bank, at randomly selected time intervals. The program in the given form has synchronization errors and logical errors. The program itself compiles correctly.

In the bank there are also 2 atm machines. Meaning at a time there are only 2 processes that can request to view the balance or can withdraw or deposit money.

Identify the critical section and synchronize the program. You would need to use P(sem) and V(sem) operations at the right places to solve the synchronization problem. You will also need to find the logical flaw if there is any and try to solve it. To use P(sem) and V(sem) include the provided "sem.h" file. Your solution should solve the following problems:

- Prevent race conditions
- Prevent a son from withdrawing money when there is no balance.
- Prevent undefined outputs like negative balance
- Prevent a process from continuously requesting access to the shared memory. Prevent unnecessary cpu cycle.
- The problem should be solved with as few semaphore variables as possible.

Submit a report showing the critical section of the code (and logical errors) and explain your solution in detail.

Marks: 15

TASK 2 **DUE: 04/27/2023**

Consider a system with 3 smoker processes and 1 agent process. Each smoker continuously rolls a cigarette and then smokes it. The smoker needs three ingredients: tobacco, paper, and matches. One of the smokers has paper, another has tobacco, and the third has matches. The agent has an infinite supply of all three materials and (randomly) places two of the ingredients on the table each time. The smoker who has the remaining ingredient then makes and smokes a cigarette, signaling the agent on completion. The agent then puts out another two of the three ingredients, and the cycle repeats.

TO DO: Write **two** programs to synchronize the agent and smoker processes: one using **semaphores** and another using **pthread libraries.**

Instructions

- Please see this link for pseudocode:

  http://www.cs.umd.edu/~hollings/cs412/s96/synch/smokers.html

- Though the description says the agent process can infinitely supply two of the three ingredients, you can assume that the agent places ingredients only a finite number of times, say for example 10.
- You need to use the “sem.h” header file in your semaphore-based solution.
- You need to include the <semaphore.h> and <pthread.h> for the thread based solution.

Marks: 15

Example Code for thread creation:

```c
/*
This is an example of how to create threads with sempaphore and mutex
implementation.
*/
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <semaphore.h>
#define THREAD_NUM 4
sem_t semaphore;
int value=1;
pthread_mutex_t buffer_mutex;
void* routine(void* args) {
    sem_wait(&semaphore);
    printf("Hello from thread %d\n", *(int*)args);
    printf("Before Increment: In thread %d values is: %d\n", *(int*)args,value);
    for(int i=0; i<1000000; i++);
    pthread_mutex_lock(&buffer_mutex); /* protecting critical section */
    value++;
    printf("After Increment: In thread %d values is: %d\n", *(int*)args,value);
    pthread_mutex_unlock(&buffer_mutex);
    sem_post(&semaphore);
    free(args);
}
int main(int argc, char *argv[]) {
    pthread_t th[THREAD_NUM];
    sem_init(&semaphore, 0, 4);
    pthread_mutex_init(&buffer_mutex, NULL);
    int i;
    for (i = 0; i < THREAD_NUM; i++) {
        int* a = malloc(sizeof(int));
        *a = i;
        if (pthread_create(&th[i], NULL, &routine, a) != 0) {
            perror("Failed to create thread");
        }
    }
    for (i = 0; i < THREAD_NUM; i++) {
        if (pthread_join(th[i], NULL) != 0) {
            perror("Failed to join thread");
        }
    }
    sem_destroy(&semaphore);
    return 0;
}
```

Submission Instructions

- All the programs MUST be clearly indented and internally documented
- Do not include executables in the zip file.
- Make sure your programs compile and run without any errors
- Save all your programs with meaningful names and zip into a single folder as: Lab6_[your last name here].zip (e.g., Lab6_Xyz.zip)
- Email your code with the subject line, "Lab6-CSC33200(L)–Class#MM2-*lastname*" (e.g., Lab6 - CSC33200(L)-Class #MM2-Xyz)
- Email: sdebnath@ccny.cuny.edu