CS105 – Computer Systems Spring 2023

Week 10: Synchronization

SOLUTION

April 3-5, 2023

1. You and a friend have decided to implement an infinite, virtual ping pong game using two threads. One thread writes "Ping!" and another writes "Pong!" The output must strictly alternate—each "Ping!" is immediately followed by a "Pong!", and *vice versa.* The program should satisfy the following properties:

   - "Ping!" goes first. "Ping!" and "Pong!" properly alternate.
   - The game goes on indefinitely. There is no possibility of deadlock.
   - There is no "busy waiting." Neither thread wastes cycles by continuously looping and looking at a variable.

   Your friend has implemented the following version:

```
int ping_count = 0;

void *ping(void* p) {
    while (1){
        if (ping_count == 0) {
            printf("Ping!\n");
            ping_count++;
        }
    }
}

void *pong(void* p) {
    while (1){
        if (ping_count == 1) {
            printf("Pong!\n");
            ping_count--;
        }
    }
}

int main() {
    pthread_t ping_tid, pong_tid;
    pthread_create(&ping_tid, NULL, ping, NULL);
    pthread_create(&pong_tid, NULL, pong, NULL);
    pthread_join(ping_tid, NULL);
    return 0;
}
```

   (a) What is wrong with this implementation?
   No synchronizated access to shared variable `ping_count`, busy waiting occurs

(b) In the space below, provide a correctly synchronized version by adding global variable(s) of the types `lock` and `cv` and rewriting the functions `ping` and `pong`. For simplicity, you may assume that the main function is correct and that the synchronization variables are automatically initialized. You should use the following notation for your synchronization: `acquire(lock)`, `release(lock)`, `wait(cv, lock)`, `signal(cv)`, `broadcast(cv)`.

```
int ping_count = 0;
lock mutex;
cv ping_turn;
cv pong_turn;


void *ping(void* p) {
    while (1) {

        acquire(mutex);

        while(ping_count == 0){
            wait(ping_turn, mutex);
        }

        printf("Ping!\n");
        ping_count++;

        signal(pong_turn);
        release(mutex);

    }
}

void *pong(void* p) {
    while (1) {

        acquire(mutex);

        while(ping_count == 1){
            wait(pong_turn, mutex);
        }

        printf("Pong!\n");
        ping_count--;

        signal(ping_turn);
        release(mutex);

    }
    }
```

2. Assume that LA county decides to one day create a functional public transit system, including a bus network. Riders come to a bus stop and wait for a bus. When the bus arrives, all the waiting riders board the bus one after another, but anyone who arrives while the bus is boarding has to wait for the next bus. The capacity of these hypothetical LA buses is 50 people; if there are more than 50 people waiting, some will have to wait for the next bus. When all the waiting riders have boarded, the bus may depart. If the bus arrives when there are no riders, it should depart immediately. Use locks and condition variables to modify the following code to enforce these constraints using the same syntax as the previous problem.

```
// global variables
int num_in_line = 0;
int num_to_board = 0;
lock bus_lock;
cv bus_arrived;
cv bus_loaded;
```

```
void bus(){
    acquire(bus_lock);

    num_bus = 0;
    if(num_in_line >= 50){
        num_to_board = 50;
    } else {
        num_to_board = num_in_line;
    }

    for(int i =0; i < num_to_board; i++){
        signal(bus_arrived);
    }

    if(num_in_line > 0){
        wait(bus_loaded, bus_lock);
    }

    release(bus_lock);

    depart();
}
```

```
void passenger(){

    acquire(bus_lock);
    num_in_line++;
    wait(bus_arrived, bus_lock);


    // board bus
    num_bus++;
    num_in_line--;


    if(num_bus == num_to_board){
        signal(bus_loaded);
    }

}
```