گزارش کار سوال سوم تمرین سری چهارم سیستم‌های عامل – محمدپویا تراشی – ۴۰۱۵۲۱۱۰۲

این سوال با الگوگیری از مسئلهٔ کلاسیک Dining Philosophers حل شده است. حل naive این مثال، همانطور که در کد تمپلیت آمده است زمان زیادی را می‌گیرد. حل اولیهٔ مسئلهٔ غذاخوردن فلاسفه نیز دچار مشکل deadlock است.

```
// For handeling right turn logic
void turn_right(int id, int direction){
    pthread_mutex_lock(&straight[leftHand(id)]);
    pthread_mutex_lock(&turnLeft[leftHand(id)]);
    pthread_mutex_lock(&turnLeft[frontSide(id)]);

    // Improve this Function
    pthread_mutex_lock(&Intersection_lock);
    switch (direction){…
    pthread_mutex_unlock(&Intersection_lock);

    pthread_mutex_unlock(&straight[leftHand(id)]);
    pthread_mutex_unlock(&turnLeft[leftHand(id)]);
    pthread_mutex_unlock(&turnLeft[frontSide(id)]);

}
```

```
// For handeling Streight logic
void go_straight(int id, int direction){
    pthread_mutex_lock(&turnLeft[rightHand(id)]);
    pthread_mutex_lock(&turnRight[rightHand(id)]);
    pthread_mutex_lock(&straight[rightHand(id)]);
    pthread_mutex_lock(&turnLeft[frontSide(id)]);
    pthread_mutex_lock(&turnLeft[leftHand(id)]);
    pthread_mutex_lock(&straight[leftHand(id)]);


    // Improve this function
    pthread_mutex_lock(&Intersection_lock);
    switch (direction){…
    pthread_mutex_unlock(&Intersection_lock);


    pthread_mutex_unlock(&turnLeft[rightHand(id)]);
    pthread_mutex_unlock(&turnRight[rightHand(id)]);
    pthread_mutex_unlock(&straight[rightHand(id)]);
    pthread_mutex_unlock(&turnLeft[frontSide(id)]);
    pthread_mutex_unlock(&turnLeft[leftHand(id)]);
    pthread_mutex_unlock(&straight[leftHand(id)]);
}
```

```
// For handeling left turn logic
void turn_left(int id, int direction){

    pthread_mutex_lock(&turnLeft[rightHand(id)]);
    pthread_mutex_lock(&turnRight[rightHand(id)]);
    pthread_mutex_lock(&straight[rightHand(id)]);
    pthread_mutex_lock(&turnLeft[frontSide(id)]);
    pthread_mutex_lock(&turnRight[frontSide(id)]);
    pthread_mutex_lock(&straight[frontSide(id)]);
    pthread_mutex_lock(&turnLeft[leftHand(id)]);
    pthread_mutex_lock(&straight[leftHand(id)]);


    // Improve this function
    pthread_mutex_lock(&Intersection_lock);
    switch (direction){…
    pthread_mutex_unlock(&Intersection_lock);


    pthread_mutex_unlock(&turnLeft[rightHand(id)]);
    pthread_mutex_unlock(&turnRight[rightHand(id)]);
    pthread_mutex_unlock(&straight[rightHand(id)]);
    pthread_mutex_unlock(&turnLeft[frontSide(id)]);
    pthread_mutex_unlock(&turnRight[frontSide(id)]);
    pthread_mutex_unlock(&straight[frontSide(id)]);
    pthread_mutex_unlock(&turnLeft[leftHand(id)]);
    pthread_mutex_unlock(&straight[leftHand(id)]);
}
```

در این راه حل برای هر یک از حرکات ممکن (گردش به راست، گردش به چپ، حرکت مستقیم) یک آرایهٔ چهارتایی mutex تعریف می‌شود و هر یک از خانه‌های این آرایه به sourceهای موجود اشاره دارند. (به طور مثال وقتی خودرویی از source ۱ می‌خواهد به راست بپیچد، turnRight[1] را قفل می‌نماید. این راه حل همانند راه حل naive برای مسئلهٔ غذا خوردن فلاسفه دچار deadlock است.

با توجه به ددلاک موجود، مسئلهٔ چهارراه بر اساس راه حل دایجسترا از مسئلهٔ غذا خوردن فیلسوف‌ها حل می‌گردد. ماشین‌ها در هر مرحله، یک حرکت تصادفی را انجام می‌دهند. در هر کدام از این حرکات، باید مشابه چهار عملیات thinking، take_forks، eat و put_down شبیه‌سازی شود. ساختار هر یک از حرکات بدین شکل می‌شود:

```
void test(int i, enum Movement proposed_state)…

// For handeling rightHand turn logic
void turn_right(int id, int direction){

    // take:
    pthread_mutex_lock(&critical_section_lock);
    state[direction] = WTRIGHT;
    test(direction, TRIGHT);
    pthread_mutex_unlock(&critical_section_lock);
    pthread_mutex_lock(&available[direction]);


    // Improve this Function
    pthread_mutex_lock(&Intersection_lock);
    switch (direction){…
    pthread_mutex_unlock(&Intersection_lock);



    // put:
    pthread_mutex_lock(&critical_section_lock);
    state[direction] = STAY;
    test(leftHand(direction), STRAIGHT);
    test(leftHand(direction), TLEFT);
    test(frontSide(direction), TLEFT);
    pthread_mutex_unlock(&critical_section_lock);
}
```

```
// For handeling Streight logic
void go_straight(int id, int direction){

    // take:
    pthread_mutex_lock(&critical_section_lock);
    state[direction] = WSTRAIGHT;
    test(direction, STRAIGHT);
    pthread_mutex_unlock(&critical_section_lock);
    pthread_mutex_lock(&available[direction]);

    // Improve this function
    pthread_mutex_lock(&Intersection_lock);
    switch (direction){…
    pthread_mutex_unlock(&Intersection_lock);



    // put:
    pthread_mutex_lock(&critical_section_lock);
    state[direction] = STAY;
    test(leftHand(direction), STRAIGHT);
    test(leftHand(direction), TLEFT);
    test(rightHand(direction), STRAIGHT);
    test(rightHand(direction), TRIGHT);
    test(rightHand(direction), TLEFT);
    test(frontSide(direction), TLEFT);
    pthread_mutex_unlock(&critical_section_lock);
}
```

```
// For handeling leftHand turn logic
void turn_left(int id, int direction){
    // think done

    // take:
    pthread_mutex_lock(&critical_section_lock);
    state[direction] = WTLEFT;
    test(direction, TLEFT);
    pthread_mutex_unlock(&critical_section_lock);
    pthread_mutex_lock(&available[direction]);


    // eat:
    // Improve this function
    pthread_mutex_lock(&Intersection_lock);
    switch (direction){…
    pthread_mutex_unlock(&Intersection_lock);

    // put:
    pthread_mutex_lock(&critical_section_lock);
    state[direction] = STAY;
    test(leftHand(direction), STRAIGHT);
    test(leftHand(direction), TLEFT);
    test(rightHand(direction), STRAIGHT);
    test(rightHand(direction), TRIGHT);
    test(rightHand(direction), TLEFT);
    test(frontSide(direction), STRAIGHT);
    test(frontSide(direction), TLEFT);
    test(frontSide(direction), TRIGHT);
    pthread_mutex_unlock(&critical_section_lock);
}
```

در کد موجود، بخش‌های take_forks، eat و put_down مشخص شده‌اند. تابع تست نیز بدین شکل است. در تمامی توابع، فقط حرکاتی که با سایر موارد conflict دارند بررسی شده‌اند.

```
 1. void test(int i, enum Movement proposed_state)
 2. {
 3.     if ((state[i] == WTLEFT && (state[rightHand(i)] != STRAIGHT && state[rightHand(i)] !=
TLEFT) && (state[frontSide(i)] != TLEFT && state[frontSide(i)] != STRAIGHT &&
state[frontSide(i)] != TRIGHT) && (state[leftHand(i)] != TLEFT && state[leftHand(i)] !=
STRAIGHT)) ||
 4.         (state[i] == WSTRAIGHT && (state[rightHand(i)] != STRAIGHT && state[rightHand(i)] !=
TLEFT && state[rightHand(i)] != TRIGHT) && (state[frontSide(i)] != TLEFT) && (state[leftHand(i)]
!= STRAIGHT && state[leftHand(i)] != TLEFT)) ||
 5.         (state[i] == WTRIGHT && (state[frontSide(i)] != TLEFT) && (state[leftHand(i)] !=
STRAIGHT && state[leftHand(i)] != TLEFT)))
 6.     {
 7.         state[i] = proposed_state;
 8.         pthread_mutex_unlock(&available[i]);
 9.     }
10.
11. }
12.
```

```
1.
```