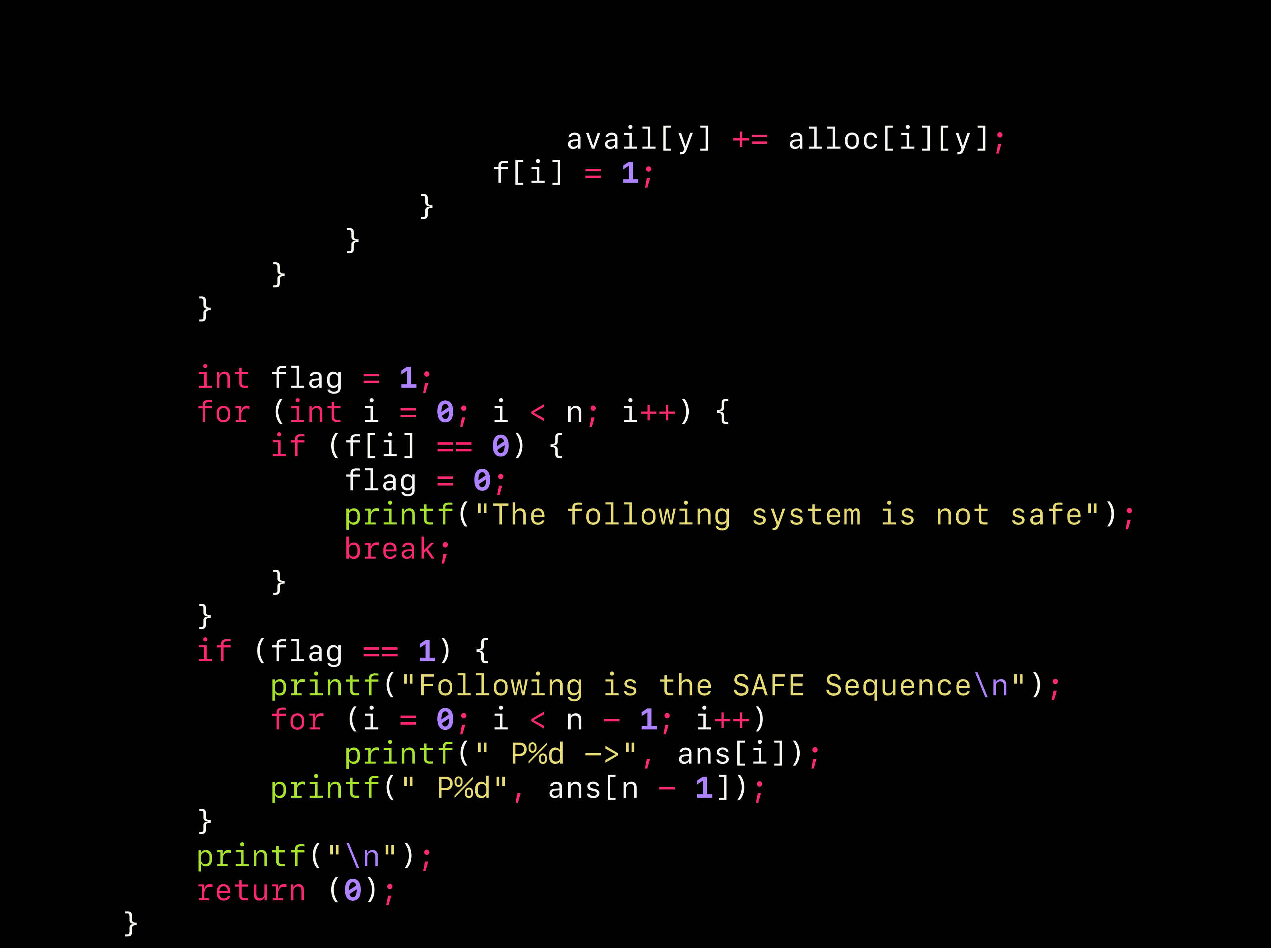
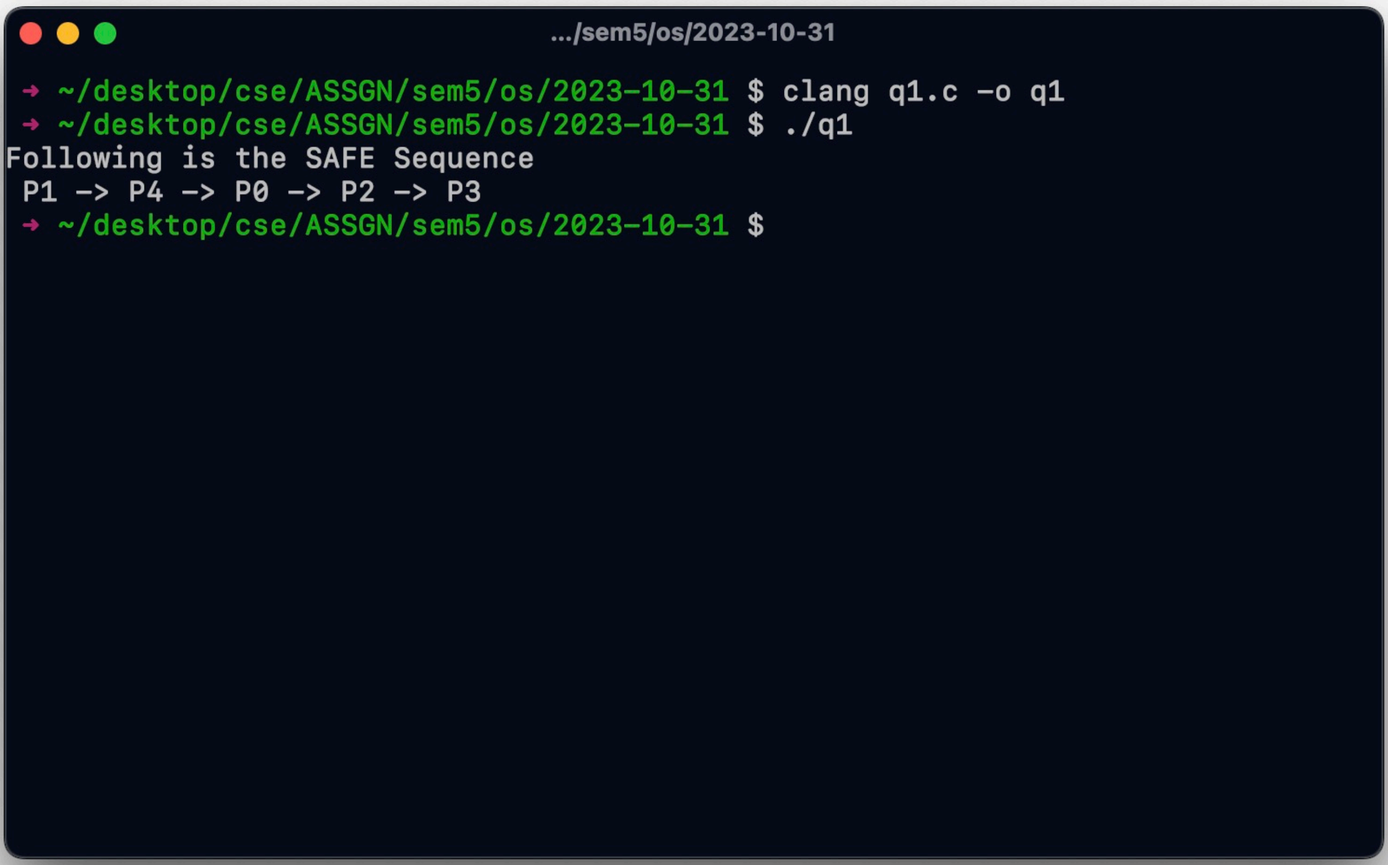
ASSIGNMENT 7
31/10/23

NAME : SHRESTH SONKAR
REGNO : 20214272
GROUP : CS5D
TOPIC : OS LAB
CODE : CS-15203

```c
// Q1 Banker's Algorithm

#include <stdio.h>

int main() {
    int n, m, i, j, k;
    n = 5;
    m = 3;
    int alloc[5][3] = {{1, 1, 2},
                       {2, 1, 2},
                       {4, 0, 1},
                       {0, 2, 0},
                       {1, 1, 2}};

    int max[5][3] = {{4, 3, 3},
                     {3, 2, 2},
                     {9, 0, 2},
                     {7, 5, 3},
                     {1, 1, 2}};

    int avail[3] = {2, 1, 0};

    int f[n], ans[n], ind = 0;
    for (k = 0; k < n; k++) {
        f[k] = 0;
    }
    int need[n][m];
    for (i = 0; i < n; i++) {
        for (j = 0; j < m; j++)
            need[i][j] = max[i][j] - alloc[i][j];
    }
    int y = 0;
    for (k = 0; k < 5; k++) {
        for (i = 0; i < n; i++) {
            if (f[i] == 0) {
                int flag = 0;
                for (j = 0; j < m; j++) {
                    if (need[i][j] > avail[j]) {
                        flag = 1;
                        break;
                    }
                }
                if (flag == 0) {
                    ans[ind++] = i;
                    for (y = 0; y < m; y++)
```

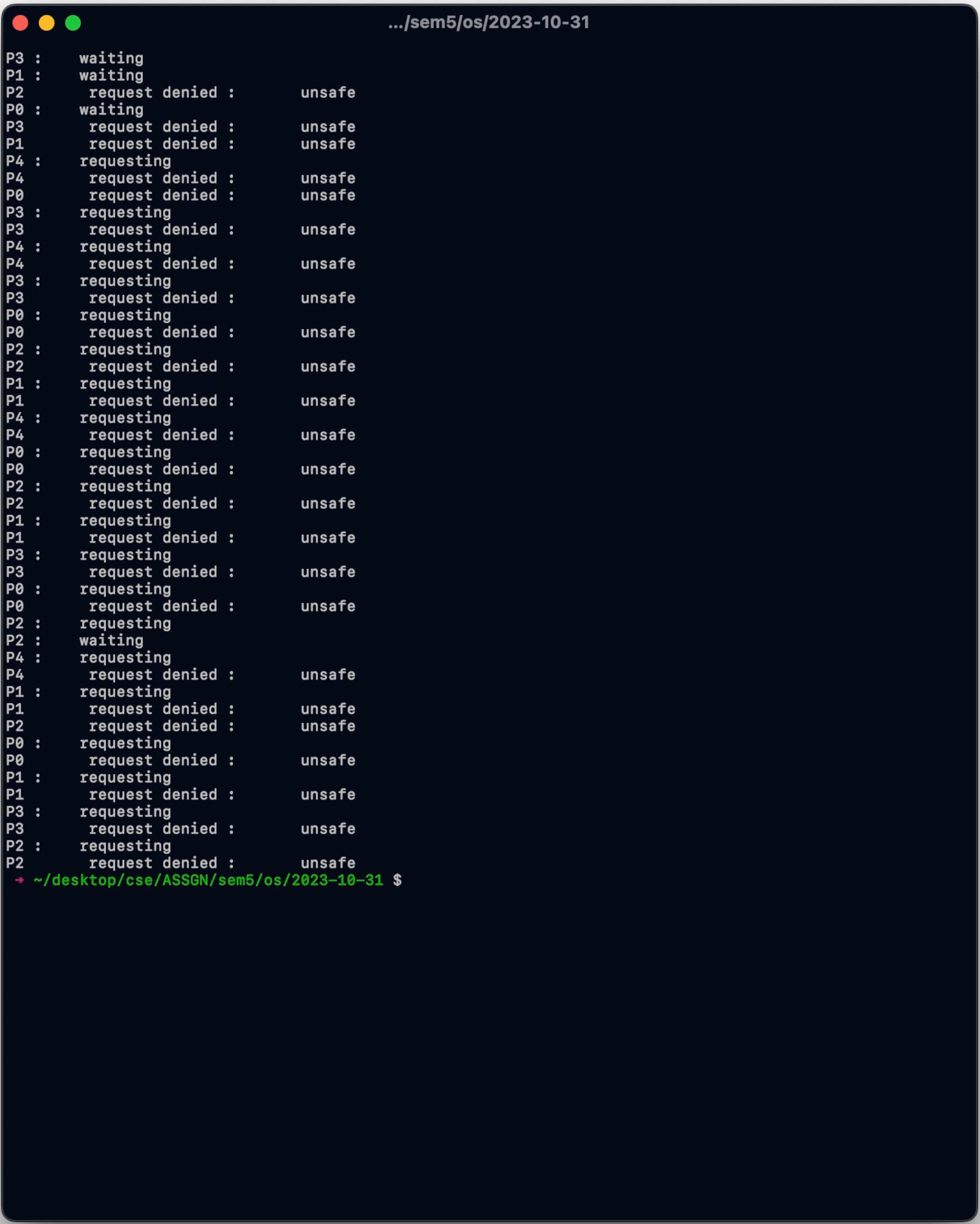


```c
                        avail[y] += alloc[i][y];
                    f[i] = 1;
                }
            }
        }
    }

    int flag = 1;
    for (int i = 0; i < n; i++) {
        if (f[i] == 0) {
            flag = 0;
            printf("The following system is not safe");
            break;
        }
    }
    if (flag == 1) {
        printf("Following is the SAFE Sequence\n");
        for (i = 0; i < n - 1; i++)
            printf(" P%d ->", ans[i]);
        printf(" P%d", ans[n - 1]);
    }
    printf("\n");
    return (0);
}
```



```
➜ ~/desktop/cse/ASSGN/sem5/os/2023-10-31 $ clang q1.c -o q1
➜ ~/desktop/cse/ASSGN/sem5/os/2023-10-31 $ ./q1
Following is the SAFE Sequence
 P1 -> P4 -> P0 -> P2 -> P3
➜ ~/desktop/cse/ASSGN/sem5/os/2023-10-31 $
```

```c
// Q2 multithreaded banker

#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <unistd.h>

#define NUM_THREADS 5
#define NUM_RESOURCES 3

int available[NUM_RESOURCES];
int allocation[NUM_THREADS][NUM_RESOURCES];
int maximum[NUM_THREADS][NUM_RESOURCES];
int need[NUM_THREADS][NUM_RESOURCES];

pthread_mutex_t mutex;

int alloc[NUM_THREADS][NUM_RESOURCES] = {{1, 1, 2},
                                         {2, 1, 2},
                                         {4, 0, 1},
                                         {0, 2, 0},
                                         {1, 1, 2}};

int max[NUM_THREADS][NUM_RESOURCES] = {{4, 3, 3},
                                       {3, 2, 2},
                                       {9, 0, 2},
                                       {7, 5, 3},
                                       {1, 1, 2}};

int avail[NUM_RESOURCES] = {2, 1, 0};

int system_is_safe() {
    int i, j, k;

    int f[NUM_THREADS], ans[NUM_THREADS], ind = 0;
    for (k = 0; k < NUM_THREADS; k++) {
        f[k] = 0;
    }
    int need[NUM_THREADS][NUM_RESOURCES];
    for (i = 0; i < NUM_THREADS; i++) {
        for (j = 0; j < NUM_RESOURCES; j++)
            need[i][j] = max[i][j] - alloc[i][j];
    }
    int y = 0;
    for (k = 0; k < NUM_THREADS; k++) {
        for (i = 0; i < NUM_RESOURCES; i++) {
            if (f[i] == 0) {
                int flag = 0;
                for (j = 0; j < NUM_RESOURCES; j++) {
                    if (need[i][j] > avail[j]) {
                        flag = 1;
                        break;
                    }
                }
                if (flag == 0) {
                    ans[ind++] = i;
                    for (y = 0; y < NUM_RESOURCES; y++)
                        avail[y] += alloc[i][y];
                    f[i] = 1;
                }
            }
        }
    }

    int flag = 1;
    for (int i = 0; i < NUM_THREADS; i++) {
        if (f[i] == 0) {
            flag = 0;
            return 0;
        }
    }
    if (flag == 1) {
        printf("Following is the SAFE Sequence\n");
        for (i = 0; i < NUM_THREADS - 1; i++)
            printf(" P%d ->", ans[i]);
        printf(" P%d", ans[NUM_THREADS - 1]);
    }
    printf("\n");
    return 1;
}

void request_resources(int thread_id) {
    pthread_mutex_lock(&mutex);
    printf("P%d : \trequesting\n", thread_id);
    int request[NUM_RESOURCES];

    for (int i = 0; i < NUM_RESOURCES; i++) {
        request[i] = rand() % (maximum[thread_id][i] -
allocation[thread_id][i] + 1);
        if (request[i] > available[i]) {
            printf("P%d : \twaiting\n", thread_id);
            pthread_mutex_unlock(&mutex);
            usleep(rand() % 1000000);
            pthread_mutex_lock(&mutex);
        }
    }

    int can_grant = 1;
    for (int i = 0; i < NUM_RESOURCES; i++) {
        if (request[i] > need[thread_id][i]) {
            can_grant = 0;
            break;
        }
    }

    if (can_grant) {
        for (int i = 0; i < NUM_RESOURCES; i++) {
            allocation[thread_id][i] += request[i];
            need[thread_id][i] -= request[i];
            available[i] -= request[i];
        }

        if (system_is_safe()) {
            printf("P%d : \tgranted\n", thread_id);
        } else {
            for (int i = 0; i < NUM_RESOURCES; i++) {
                allocation[thread_id][i] -= request[i];
                need[thread_id][i] += request[i];
                available[i] += request[i];
            }
            printf("P%d\t request denied : \tunsafe\n", 
thread_id);
        }
    } else {
        printf("P%d\t request denied : \tinsufficient 
resources\n", thread_id);
    }

    pthread_mutex_unlock(&mutex);
}

void release_resources(int thread_id) {
    pthread_mutex_lock(&mutex);
    for (int i = 0; i < NUM_RESOURCES; i++) {
        available[i] += allocation[thread_id][i];
        allocation[thread_id][i] = 0;
        need[thread_id][i] = maximum[thread_id][i];
    }

    pthread_mutex_unlock(&mutex);
}

void *thread_function(void *arg) {
    int thread_id = *((int *) arg);
    for (int i = 0; i < 5; i++) {
        usleep(rand() % 1000000);
        request_resources(thread_id);
        usleep(rand() % 1000000);
        release_resources(thread_id);
    }

    return NULL;
}

int main() {
    srand(time(NULL));
    pthread_mutex_init(&mutex, NULL);

    for (int i = 0; i < NUM_THREADS; i++) {
        for (int j = 0; j < NUM_RESOURCES; j++) {
            maximum[i][j] = max[i][j];
            allocation[i][j] = alloc[i][j];
            need[i][j] = max[i][j] - alloc[i][j];
        }
    }

    pthread_t threads[NUM_THREADS];
    int thread_args[NUM_THREADS];

    for (int i = 0; i < NUM_THREADS; i++) {
        thread_args[i] = i;
        pthread_create(&threads[i], NULL,
thread_function, &thread_args[i]);
    }

    for (int i = 0; i < NUM_THREADS; i++) {
        pthread_join(threads[i], NULL);
    }

    pthread_mutex_destroy(&mutex);
    return 0;
}
```



```
P3 :    waiting
P1 :    waiting
P2       request denied :       unsafe
P0 :    waiting
P3       request denied :       unsafe
P1       request denied :       unsafe
P4 :    requesting
P4       request denied :       unsafe
P0       request denied :       unsafe
P3 :    requesting
P3       request denied :       unsafe
P4 :    requesting
P4       request denied :       unsafe
P3 :    requesting
P3       request denied :       unsafe
P0 :    requesting
P0       request denied :       unsafe
P2 :    requesting
P2       request denied :       unsafe
P1 :    requesting
P1       request denied :       unsafe
P4 :    requesting
P4       request denied :       unsafe
P0 :    requesting
P0       request denied :       unsafe
P2 :    requesting
P2       request denied :       unsafe
P1 :    requesting
P1       request denied :       unsafe
P3 :    requesting
P3       request denied :       unsafe
P0 :    requesting
P0       request denied :       unsafe
P2 :    requesting
P2 :    waiting
P4 :    requesting
P4       request denied :       unsafe
P1 :    requesting
P1       request denied :       unsafe
P2       request denied :       unsafe
P0 :    requesting
P0       request denied :       unsafe
P1 :    requesting
P1       request denied :       unsafe
P3 :    requesting
P3       request denied :       unsafe
P2 :    requesting
P2       request denied :       unsafe
 ➜ ~/desktop/cse/ASSGN/sem5/os/2023-10-31 $
```