# OS161 운영체제 환경에서의 데드락 문제

## 1. 서론

데드락(교착 상태)은 집합에 속한 각 프로세스가 다른 프로세스의 완료를 기다리다가 서로 맞물려서 진행 불가 상태가 발생하는 것으로, 서로가 상대방의 작업이 끝나기 만을 기다리고 있기 때문에 결과적으로 아무것도 완료되지 못하는 상태이다. 이에 따라 데드락은 영구적이고 효율적인 해결 방법이 존재하지 않으며, 결과적으로 시스템의 정지를 유발한다.

데드락은 다음 4종류의 필요 조건을 충족시켜야 발생한다.

1) 상호배제: 프로세스들이 필요로 하는 자원에 대해 배타적인 통제권을 요구한다.
2) 점유대기: 프로세스가 할당된 자원을 가진 상태에서 다른 자원을 기다린다.
3) 비선점: 프로세스가 어떤 자원의 사용을 끝낼 때까지 그 자원을 뺏을 수 없다.
4) 순환대기: 각 프로세스는 순환적으로 다음 프로세스가 요구하는 자원을 가지고 있다.

## 2. 데드락의 해결 방법

데드락에 대한 해결 방법은 아래 4종류가 존재한다.

1) 데드락 예방: 시스템에서 데드락이 발생할 수 있는 여지가 없도록 설계한다. 이를 위해서는 각각의 필요 조건에 대해 OS가 상호 배제를 지원해야 하며, 모든 자원 할당이 이루어질 수 있을 때까지 요청을 막고, 자원을 가진 프로세스의 요청이 거부되면 해당 프로세스는 자원을 되돌려 놔야 하며, 선형적인 순서를 정하고 차례대로 할당하는 작업이 최소 하나는 달성되어야 한다.
2) 데드락 회피: 각 요청을 OS가 직접 분석하여 데드락이 발생할 가능성이 존재하는지 확인한다.
3) 데드락 발견: 데드락이 항상 발생할 것을 가정하고 현재 시스템에서 어느 부분에 데드락이 발생했는지 탐색하며, 시스템 상태를 자원 할당 측면에서만 확인하고 현재 상태에 따라 할당한다.
4) 데드락 복구: 데드락이 사라질 때까지 특정 작업을 수행한다. 여기에는 모든 데드락 걸린 프로세스를 종료하거나, 프로세스에 중간 포인트를 걸어 이전 프로세스로 되돌아가거나, 프로세스를 순차적으로 종료하거나, 자원을 선점하는 방법이 존재한다.

## 3. 데드락 해결 알고리즘에 대한 구상

본 과제에서는 교차로와 그 내부의 교통을 가정하고 세마포어를 사용함으로써 데드락의 상황을 방지하는 하는 것을 목표로 하고 있다. 교차로를 통과하는 자동차는 직진, 우회전, 좌회전의 통과를 목적으로 진입하게 되며, 교차로를 사분면으로 나누어 통과는 교차점의 부분들을 통한 진행으로 표현된다. 교차로의 같은 부분에는 2대 이상의 자동차가 동시에 존재할 수 없으며, 다른 방향에서 오는 트래픽이 무한정 기다리는 꼬리 물기를 하지 못하도록 하고, 각 차량은 차량 번호, 접근 방향 및 목적지 방향을 나타내는 출력을 교차로에 접근, 진입 및 떠날 때 인쇄해야 한다.

주어진 목적을 달성하기 위해 구상한 알고리즘은 다음과 같다.

1) 신호등: 이 방법은 프로세스에 순차적으로 실행 순서를 부여하는 방법으로, 우선순위에 상관없이 실행되므로 기아 상태를 개선할 것으로 예측된다. 이 방법은 데드락을 발생시키지 않게 하는 데드락

예방에 속한다.

2) 은행원 알고리즘 이용: 이 방법은 데드락에 빠질 가능성을 OS가 검사한 후 그럴 가능성이 없다고 판단되면 자원을 할당하는 방법이다. 이 방법은 각 요청의 데드락 가능성을 분석하는 데드락 회피에 속한다.

3) 고가도로: 이 방법은 데드락이 발생할 시 특정 프로세스를 고가도로(별개의 메모리)로 이동시켜 데드락과 관계없이 실행하게 하는 방법이다. 이 방법은 데드락 발생 후 특정 작업을 수행하는 데드락 복구에 속한다.

본 문제에서는 해결 방법으로 신호등 방법을 택하였으며, 이를 상세히 구현하기 위해 이미 존재하는 코드를 기반으로 최적화를 진행하였다.

## 4. 알고리즘 코드 구축

1) 교차로 구역을 위한 세마포어와 메시지 출력을 위한 스핀락을 정의한다.

```
// Directions, cars in intersection
static struct semaphore *NW=0;
static struct semaphore *NE=0;
static struct semaphore *SW=0;
static struct semaphore *SE=0;
static struct semaphore *TSIGN[4]={0};  // traffic sign
static struct spinlock *msg_lock=0;     // messager lock
```

2) 초기화 및 정리 함수인 init_synch()와 free_synch()를 정의한다. init_synch()는 시뮬레이션에 필요한 세마포어와 스핀락을 초기화하고, free_synch()는 시뮬레이션 후 세마포어와 스핀락을 제거한다.

```
static void init_synch(void) {
    if (!donesem) {
        donesem = sem_create("donesem", 0);
        if (!donesem) { panic("synchtest: sem_create failed₩n"); }
    }
    if (!NW) {
        NW = sem_create("NW", 1);
        if (!NW) { panic("synchtest: sem_create failed₩n"); }
    }
    if (!NE) {
        NE = sem_create("NE", 1);
        if (!NE) { panic("synchtest: sem_create failed₩n"); }
    }
    if (!SW) {
        SW = sem_create("SW", 1);
        if (!SW) { panic("synchtest: sem_create failed₩n"); }
    }
    if (!SE) {
        SE = sem_create("SE", 1);
        if (!SE) { panic("synchtest: sem_create failed₩n"); }
    }
    for (int i = 0; i < 4; i++) {
        TSIGN[i] = sem_create("TSIGN", 0);
        if (!TSIGN[i])
            panic("synchtest: sem_create failed₩n");
    }
    spinlock_init(&msg_lock);
}
```

```
static void free_synch(void) {
    spinlock_cleanup(&msg_lock);
    for (int i = 0; i < 4; i++) {
        if (TSIGN[i]) {
            sem_destroy(TSIGN[i]);
            TSIGN[i] = 0;
        }
    }
    if (NW) { sem_destroy(NW); NW = 0; }
    if (NE) { sem_destroy(NE); NE = 0; }
    if (SW) { sem_destroy(SW); SW = 0; }
    if (SE) { sem_destroy(SE); SE = 0; }
    if (donesem) { sem_destroy(donesem); donesem = 0; }
}
```

3) 메시지 출력 함수 message_wait()와 message_move()를 정의한다. message_wait()는 자동차가 교차로에 들어가기를 기다리고 있음을 나타내는 메시지를 출력하며, message_move()는 자동차가 교차로의 한 구역에서 다른 구역으로 이동할 때의 메시지를 출력한다.

```
static void message_wait(int num, char start, int turn) {
    static const char *turn_str[3] = {
        "go straight",
        "turn right",
        "turn left",
    };

    message("#%02d: waiting in %c to %s\n", num, start, turn_str[turn]);
}
static void message_move(int num, const char *start, const char *prev, const char *curr, const char *end) {
    if (!strcmp(start, prev)) { // enter to intersection
        message("#%02d: enter %2s -> %2s (start:%s / end:%s)\n", num, prev, curr, start, end);
    } else if (!strcmp(curr, end)) { // exit from intersection
        message("#%02d: exit  %2s <- %2s (start:%s / end:%s)\n", num, end, prev, start, end);
    } else { // move in intersection
        message("#%02d: move  %2s -> %2s (start:%s / end:%s)\n", num, prev, curr, start, end);
    }
}
```

4) 자동차의 이동을 보여주는 함수들을 정의한다. gostraight_proc()는 교차로를 직진할 때의 절차를 처리하고, leftturn_proc()는 좌회전할 때의 절차를 처리하며, rightturn_proc()는 우회전할 때의 절차를 처리한다. 그리고 gostraight(), leftturn(), rightturn()를 사용하여 자동차의 시작 방향에 따라 적절한 절차를 호출한다.

```
static void gostraight_proc(int num, const char *start, struct semaphore *step1, struct semaphore *step2, const char *end) {
    P(step1);
    message_move(num, start, start, step1->sem_name, end);
    P(step2);
    message_move(num, start, step1->sem_name, step2->sem_name, end);
    V(step1);
    message_move(num, start, step2->sem_name, end, end);
    V(step2);
}

static void gostraight(int num, char start) {
    switch(start) {
        case 'N':
            gostraight_proc(num, "N", NW, SW, "S");
            break;
        case 'E':
            gostraight_proc(num, "E", NE, NW, "W");
            break;
        case 'S':
            gostraight_proc(num, "S", SE, NE, "N");
            break;
        case 'W':
            gostraight_proc(num, "W", SW, SE, "E");
            break;
        default:
            break;
    }
}
```

```
static void leftturn_proc(int num, const char *start, struct semaphore *step1, struct semaphore *step2, struct semaphore *step3, const char *dest) {
    P(step1);
    message_move(num, start, start, step1->sem_name, end);
    P(step2);
    message_move(num, start, step1->sem_name, step2->sem_name, end);
    V(step1);
    P(step3);
    message_move(num, start, step2->sem_name, step3->sem_name, end);
    V(step2);
    message_move(num, start, step3->sem_name, end, end);
    V(step3);
}

static void leftturn(int num, char start) {
    switch(start) {
        case 'N':
            leftturn_proc(num, "N", NW, SW, SE, "E");
            break;
        case 'E':
            leftturn_proc(num, "E", NE, NW, SW, "S");
            break;
        case 'S':
            leftturn_proc(num, "S", SE, NE, NW, "W");
            break;
        case 'W':
            leftturn_proc(num, "W", SW, SE, NE, "N");
            break;
        default:
            break;
    }
}
```

```
static void rightturn_proc(int num, const char *start, struct semaphore *step, const char *end) {
    P(step);
    message_move(num, start, start, step->sem_name, end);
    message_move(num, start, step->sem_name, end, end);
    V(step);
}

static void rightturn(int num, char start) {
    switch(start) {
        case 'N':
            rightturn_proc(num, "N", NW, "W");
            break;
        case 'E':
            rightturn_proc(num, "E", NE, "N");
            break;
        case 'S':
            rightturn_proc(num, "S", SE, "E");
            break;
        case 'W':
            rightturn_proc(num, "W", SW, "S");
            break;
        default:
            break;
    }
}
```

5) 교통 신호 스레드 car_thread()를 정의한다. car_thread()에서 각 자동차는 무작위로 시작 위치와 회전 방향을 결정하고, 대기 메시지를 출력한 후 선택된 방향에 따라 교차로를 이동한다.

```
// thread for car
static void car_thread(void *junk, unsigned long num) {
    (void)junk;

    static char start_pos[4] = {'N', 'E', 'S', 'W'};
    int start = random() % 4; // random start position
    int turn = random() % 3;  // 0: straight, 1: right turn, 2: left turn

    message_wait(num, position[start], turn);
    P(TSIGN[start]);

    switch (turn) {
    case 0:
        gostraight(num, position[start]);
        break;
    case 1:
        rightturn(num, position[start]);
        break;
    case 2:
        leftturn(num, position[start]);
        break;
    default:
        break;
    }

    V(donesem);
}
```

6) 교통 신호 스레드 tsign_thread()를 정의한다. tsign_thread()는 교통 신호 변화를 시뮬레이션 하며, 네 방향을 순환하며 각 방향에서 자동차가 진행할 수 있도록 하고, 실제 교통과 유사하게 지연을 도입하였다.

```
#define DELAY_TSIGN 10
static bool run_tsign = true;

// thread for traffic sign
static void tsign_thread(void *junk, unsigned long dir) {
    (void)junk;

    static char position[4] = {'N', 'E', 'S', 'W'};
    while (run_tsign) {
        message("Change Traffic Sign: %c₩n", position[dir]);
        V(TSIGN[dir]);
        V(TSIGN[dir]);
        V(TSIGN[dir]);
        for (int i = 0; i < DELAY_TSIGN; i++)
            thread_yield();  // delay for change
        dir = (dir + 1) % 4; // rotate 4 direction
    }
}
```

7) 테스트 수행 함수 semtest()를 정의한다.

```
int semtest(int nargs, char **args) {
    (void)nargs;
    (void)args;

    init_synch();
    for (int i = 0; i < NTHREADS; i++) {
        int result = thread_fork("semtest", NULL, car_thread, NULL, i);
        if (result) panic("semtest: thread_fork failed: %s₩n", strerror(result));
    }

    int result = thread_fork("semtest", NULL, tsign_thread, NULL, 0);
    if (result) panic("semtest: thread_fork failed: %s₩n", strerror(result));

    for (int i = 0; i < NTHREADS; i++) // wait for all car_thread done
        P(donesem);
    run_tsign = false; // stop tsign_thread
    free_synch();

    kprintf("Semaphore test done.₩n");
    return 0;
}
```

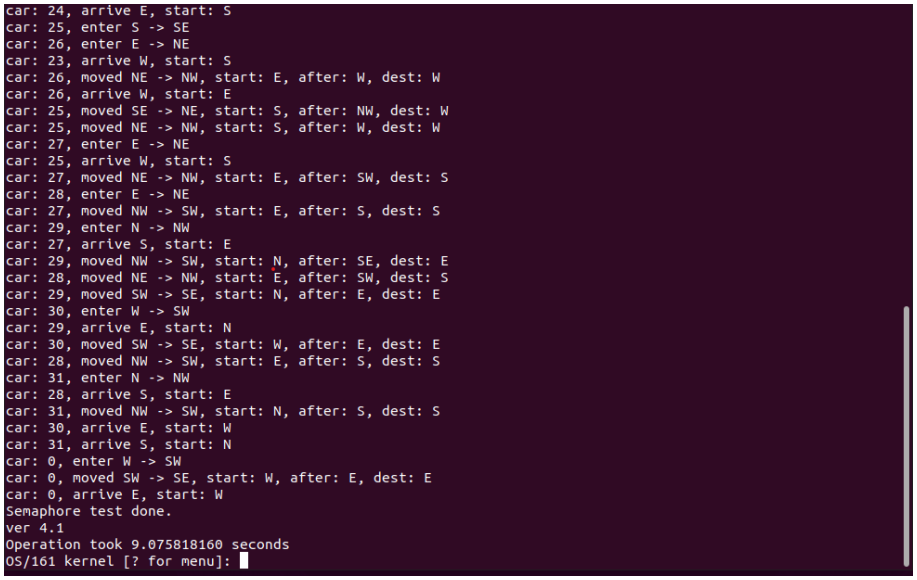
작성된 코드에 따라 테스트를 진행하면 다음과 같이 수행될 것이다.

1) init_synch() 함수가 호출되어 세마포어와 스핀락을 설정한다.
2) tsign_thread()가 시작되어 교통 신호를 계속 변경하여 한 방향씩 자동차가 진행될 수 있게 한다.
3) 다수의 car_thread() 인스턴스가 시작되어 각 자동차가 교차로를 통과하는 것을 시뮬레이션 한다. 각 자동차는 시작 방향과 회전 유형을 결정하고, 신호를 기다린 후 세마포어를 사용하여 충돌 없이 교차로를 통과한다.
4) 각 방향의 세마포어는 자동차가 신호를 따르고 충돌을 피하며 교차로를 안전하게 통과하도록 보장하며, donesem 세마포어는 시뮬레이션이 완료되었음을 신호한다.
5) 모든 자동차가 교차로를 통과하고 시뮬레이션이 완료되면 free_synch() 함수가 호출되어 모든 동기화 프리미티브를 정리한다.

## 5. 실행 결과 및 분석

작성된 알고리즘의 실행 결과는 다음과 같다.

```
cpu0: MIPS/161 (System/161 2.x) features 0x0
OS/161 kernel [? for menu]: sy1
Starting semaphore test...
If this hangs, it's broken: ok
car: 13, waiting in N to turn right
car: 1, waiting in E to turn left
car: 2, waiting in S to turn left
car: 3, waiting in E to go straight
car: 4, waiting in S to turn right
car: 5, waiting in N to turn left
car: 6, waiting in S to turn right
car: 7, waiting in W to go straight
car: 8, waiting in W to turn left
car: 9, waiting in N to turn right
car: 10, waiting in S to go straight
car: 11, waiting in S to turn left
car: 12, waiting in S to turn left
car: 14, waiting in S to turn right
car: 15, waiting in E to go straight
car: 16, waiting in W to turn left
car: 17, waiting in W to turn right
car: 18, waiting in S to turn right
car: 19, waiting in W to go straight
car: 20, waiting in N to turn right
car: 21, waiting in N to go straight
car: 22, waiting in E to turn right
car: 23, waiting in S to turn left
car: 24, waiting in S to turn right
car: 25, waiting in S to turn left
car: 26, waiting in E to go straight
car: 27, waiting in E to turn left
car: 28, waiting in E to turn left
car: 29, waiting in N to turn left
car: 30, waiting in W to go straight
car: 31, waiting in N to go straight
car: 0, waiting in W to go straight
car: 13, enter N -> NW
car: 1, enter E -> NE
car: 2, enter S -> SE
car: 13, arrive W, start: N
car: 1, moved NE -> NW, start: E, after: SW, dest: S
car: 2, moved SE -> NE, start: S, after: NW, dest: W
car: 1, moved NW -> SW, start: E, after: S, dest: S
car: 2, moved NE -> NW, start: S, after: W, dest: W
car: 3, enter E -> NE
car: 2, arrive W, start: S
car: 3, moved NE -> NW, start: E, after: W, dest: W
car: 4, enter S -> SE
car: 1, arrive S, start: E
car: 3, arrive W, start: E
car: 5, enter N -> NW
car: 4, arrive E, start: S
car: 4, arrive E, start: S
car: 6, enter S -> SE
car: 5, moved NW -> SW, start: N, after: SE, dest: E
car: 6, arrive E, start: S
car: 5, moved SW -> SE, start: N, after: E, dest: E
car: 7, enter W -> SW
car: 5, arrive E, start: N
car: 7, moved SW -> SE, start: W, after: E, dest: E
car: 8, enter W -> SW
car: 7, arrive E, start: W
car: 8, moved SW -> SE, start: W, after: NE, dest: N
car: 9, enter N -> NW
car: 8, moved SE -> NE, start: W, after: N, dest: N
car: 10, enter S -> SE
car: 8, arrive N, start: W
car: 10, moved SE -> NE, start: S, after: N, dest: N
car: 11, enter S -> SE
car: 10, arrive N, start: S
car: 11, moved SE -> NE, start: S, after: NW, dest: W
car: 12, enter S -> SE
car: 9, arrive W, start: N
car: 11, moved NE -> NW, start: S, after: W, dest: W
car: 12, moved SE -> NE, start: S, after: NW, dest: W
car: 14, enter S -> SE
car: 11, arrive W, start: S
car: 12, moved NE -> NW, start: S, after: W, dest: W
car: 15, enter E -> NE
car: 12, arrive W, start: S
car: 15, moved NE -> NW, start: E, after: W, dest: W
car: 16, enter W -> SW
car: 14, arrive E, start: S
car: 16, moved SW -> SE, start: W, after: NE, dest: N
car: 17, enter W -> SW
car: 16, moved SE -> NE, start: W, after: N, dest: N
car: 15, arrive W, start: E
car: 18, enter S -> SE
car: 16, arrive N, start: W
car: 17, arrive S, start: W
car: 19, enter W -> SW
car: 20, enter N -> NW
car: 18, arrive E, start: S
car: 19, moved SW -> SE, start: W, after: E, dest: E
car: 20, arrive W, start: N
car: 21, enter N -> NW
car: 22, enter E -> NE
car: 19, arrive E, start: W
car: 23, enter S -> SE
car: 22, arrive N, start: E
car: 23, moved SE -> NE, start: S, after: NW, dest: W
car: 24, enter S -> SE
car: 21, moved NW -> SW, start: N, after: S, dest: S
car: 23, moved NE -> NW, start: S, after: W, dest: W
car: 21, arrive S, start: N
```




초기 출력: car n, waiting int X to ~ 에서는 차량들이 교차로의 특정 방향에서 대기하고 있는 모습이 나타난다. 각 차량은 교차로의 특정 방향에서 직진/우회전/좌회전을 하기 위해 대기 중이다. 교차로 진입 및 이동: car: n, enter ~ / car: n, moved ~ 에서는 각 차량이 교차로에 진입하여 이동하는 과정이 기록되어 있다. 차량들은 교차로에 진입한 후, 교차로 내부에서 이동 경로를 따라 다음 위치로 이동한다. 이 과정은 신호등 tsign에 의하여 순차적으로 실행된다. 목적지 도착: car: n arrive x ~ 에서는 각 차량이 교차로를 통과하여 최종 목적지에 도착한 상태를 나타낸다.

이 테스트에서 차량들이 교차로에서 무한히 대기하지 않고 순차적으로 이동하고 목적지에 도착하는 것을 확인할 수 있었으며, 각 차량의 진입, 이동, 도착 상태가 정상적으로 기록된 것을 통해 세마포어가 올바르게 작동했음을 알 수 있다. 결과적으로 해당 로그를 통해 OS161 환경에서 세마포어를 사용하여 교차로에서 차량의 데드락을 방지하는 테스트가 성공적으로 완료됨이 확인되었다. 각 차량이 교차로에 진입하고 직진/좌회전/우회전 방향으로 이동하며 목적지에 도착하는 과정과, 이 과정이 꼬리물기 없이 신호등에 따라 순차적으로 해결되었음이 출력되었으며, 이를 통해 세마포어가 올바르게 작동하였음을 확인할 수 있었다.

## 6. 소스코드

synchtest.c

```
#include <types.h>
#include <lib.h>
#include <clock.h>
#include <thread.h>
#include <synch.h>
#include <test.h>

#define NSEMLOOPS     63
#define NLOCKLOOPS    120
#define NCVLOOPS      5
#define NTHREADS      32

static volatile unsigned long testval1;
static volatile unsigned long testval2;
static volatile unsigned long testval3;
static struct semaphore *testsem;
static struct lock *testlock;
static struct cv *testcv;
static struct semaphore *donesem;
```

```
static void inititems(void) {
	if (testsem==NULL) {
		testsem = sem_create("testsem", 2);
		if (testsem == NULL) {
			panic("synchtest: sem_create failed\n");
		}
	}
	if (testlock==NULL) {
		testlock = lock_create("testlock");
		if (testlock == NULL) {
			panic("synchtest: lock_create failed\n");
		}
	}
	if (testcv==NULL) {
		testcv = cv_create("testlock");
		if (testcv == NULL) {
			panic("synchtest: cv_create failed\n");
		}
	}
	if (donesem==NULL) {
		donesem = sem_create("donesem", 0);
		if (donesem == NULL) {
			panic("synchtest: sem_create failed\n");
		}
	}
}

/*****************************************************************/

// Directions, cars in intersection
static struct semaphore *NW=0;
static struct semaphore *NE=0;
static struct semaphore *SW=0;
static struct semaphore *SE=0;
static struct semaphore *TSIGN[4]={0};  // traffic sign
static struct spinlock *msg_lock=0;     // messager lock

static void init_synch(void) {
	if (!donesem) {
		donesem = sem_create("donesem", 0);
		if (!donesem) { panic("synchtest: sem_create failed\n"); }
	}
	if (!NW) {
		NW = sem_create("NW", 1);
		if (!NW) { panic("synchtest: sem_create failed\n"); }
	}
	if (!NE) {
		NE = sem_create("NE", 1);
		if (!NE) { panic("synchtest: sem_create failed\n"); }
	}
	if (!SW) {
		SW = sem_create("SW", 1);
		if (!SW) { panic("synchtest: sem_create failed\n"); }
	}
	if (!SE) {
		SE = sem_create("SE", 1);
		if (!SE) { panic("synchtest: sem_create failed\n"); }
	}
	for (int i = 0; i < 4; i++) {
		TSIGN[i] = sem_create("TSIGN", 0);
		if (!TSIGN[i])
		    panic("synchtest: sem_create failed\n");
	}
	spinlock_init(&msg_lock);
}
```

```
static void free_synch(void) {
    spinlock_cleanup(&msg_lock);
    for (int i = 0; i < 4; i++) {
        if (TSIGN[i]) {
            sem_destroy(TSIGN[i]);
            TSIGN[i] = 0;
        }
    }
    if (NW) { sem_destroy(NW); NW = 0; }
    if (NE) { sem_destroy(NE); NE = 0; }
    if (SW) { sem_destroy(SW); SW = 0; }
    if (SE) { sem_destroy(SE); SE = 0; }
    if (donesem) { sem_destroy(donesem); donesem = 0; }
}

static void message_wait(int num, char start, int turn) {
    static const char *turn_str[3] = {
        "go straight",
        "turn right",
        "turn left",
    };

    message("#%02d: waiting in %c to %s\n", num, start, turn_str[turn]);
}

static void message_move(int num, const char *start, const char *prev, const
char *curr, const char *end) {
    if (!strcmp(start, prev)) { // enter to intersection
        message("#%02d: enter %2s -> %2s (start:%s / end:%s)\n", num, prev, curr,
start, end);
    } else if (!strcmp(curr, end)) { // exit from intersection
        message("#%02d: exit  %2s <- %2s (start:%s / end:%s)\n", num, end, prev,
start, end);
    } else { // move in intersection
        message("#%02d: move  %2s -> %2s (start:%s / end:%s)\n", num, prev, curr,
start, end);
    }
}

static void gostraight_proc(int num, const char *start, struct semaphore *step1,
semaphore *step2, const char *end) {
    P(step1);
    message_move(num, start, start, step1->sem_name, end);
    P(step2);
    message_move(num, start, step1->sem_name, step2->sem_name, end);
    V(step1);
    message_move(num, start, step2->sem_name, end, end);
    V(step2);
}

static void gostraight(int num, char start) {
    switch(start) {
        case 'N':
            gostraight_proc(num, "N", NW, SW, "S");
            break;
        case 'E':
            gostraight_proc(num, "E", NE, NW, "W");
            break;
        case 'S':
            gostraight_proc(num, "S", SE, NE, "N");
            break;
        case 'W':
            gostraight_proc(num, "W", SW, SE, "E");
            break;
```

```
        default:
            break;
        }
}

static void leftturn_proc(int num, const char *start, struct semaphore *step1,
 semaphore *step2, struct semaphore *step3, const char *dest) {
    P(step1);
    message_move(num, start, start, step1->sem_name, end);
    P(step2);
    message_move(num, start, step1->sem_name, step2->sem_name, end);
    V(step1);
    P(step3);
    message_move(num, start, step2->sem_name, step3->sem_name, end);
    V(step2);
    message_move(num, start, step3->sem_name, end, end);
    V(step3);
}

static void leftturn(int num, char start) {
    switch(start) {
        case 'N':
            leftturn_proc(num, "N", NW, SW, SE, "E");
            break;
        case 'E':
            leftturn_proc(num, "E", NE, NW, SW, "S");
            break;
        case 'S':
            leftturn_proc(num, "S", SE, NE, NW, "W");
            break;
        case 'W':
            leftturn_proc(num, "W", SW, SE, NE, "N");
            break;
        default:
            break;
        }
}

static void rightturn_proc(int num, const char *start, struct semaphore *step,
char *end) {
    P(step);
    message_move(num, start, start, step->sem_name, end);
    message_move(num, start, step->sem_name, end, end);
    V(step);
}

static void rightturn(int num, char start) {
    switch(start) {
        case 'N':
            rightturn_proc(num, "N", NW, "W");
            break;
        case 'E':
            rightturn_proc(num, "E", NE, "N");
            break;
        case 'S':
            rightturn_proc(num, "S", SE, "E");
            break;
        case 'W':
            rightturn_proc(num, "W", SW, "S");
            break;
        default:
            break;
        }
}
```

```
// thread for car
static void car_thread(void *junk, unsigned long num) {
      (void)junk;

        static char start_pos[4] = {'N', 'E', 'S', 'W'};
        int start = random() % 4; // random start position
        int turn = random() % 3;  // 0: straight, 1: right turn, 2: left turn

        message_wait(num, position[start], turn);
        P(TSIGN[start]);

        switch (turn) {
           case 0:
               gostraight(num, position[start]);
               break;
           case 1:
               rightturn(num, position[start]);
               break;
           case 2:
               leftturn(num, position[start]);
               break;
           default:
               break;
        }

        V(donesem);
}

#define DELAY_TSIGN 10
static bool run_tsign = true;

// thread for traffic sign
static void tsign_thread(void *junk, unsigned long dir) {
    (void)junk;

    static char position[4] = {'N', 'E', 'S', 'W'};
    while (run_tsign) {
        message("Change Traffic Sign: %c\n", position[dir]);
        V(TSIGN[dir]);
        V(TSIGN[dir]);
        V(TSIGN[dir]);
        for (int i = 0; i < DELAY_TSIGN; i++)
            thread_yield();  // delay for change
        dir = (dir + 1) % 4; // rotate 4 direction
    }
}

int semtest(int nargs, char **args) {
        (void)nargs;
        (void)args;

        init_synch();
        for (int i = 0; i < NTHREADS; i++) {
                int result = thread_fork("semtest", NULL, car_thread, NULL, i);
                if (result) panic("semtest: thread_fork failed: %s\n",
strerror(result));
        }

        int result = thread_fork("semtest", NULL, tsign_thread, NULL, 0);
        if (result) panic("semtest: thread_fork failed: %s\n", strerror(result));

        for (int i = 0; i < NTHREADS; i++) // wait for all car_thread done
                P(donesem);
           run_tsign = false; // stop tsign_thread
        free_synch();
```

```
	kprintf("Semaphore test done.\n");
	return 0;
}

	/*****************************************************************/

static void fail(unsigned long num, const char *msg) {
	kprintf("thread %lu: Mismatch on %s\n", num, msg);
	kprintf("Test failed\n");

	lock_release(testlock);

	V(donesem);
	thread_exit();
}

static void locktestthread(void *junk, unsigned long num) {
	int i;
	(void)junk;

	for (i=0; i<NLOCKLOOPS; i++) {
		lock_acquire(testlock);
		testval1 = num;
		testval2 = num*num;
		testval3 = num%3;

		if (testval2 != testval1*testval1) {
			fail(num, "testval2/testval1");
		}

		if (testval2%3 != (testval3*testval3)%3) {
			fail(num, "testval2/testval3");
		}

		if (testval3 != testval1%3) {
			fail(num, "testval3/testval1");
		}

		if (testval1 != num) {
			fail(num, "testval1/num");
		}

		if (testval2 != num*num) {
			fail(num, "testval2/num");
		}

		if (testval3 != num%3) {
			fail(num, "testval3/num");
		}

		lock_release(testlock);
	}
	V(donesem);
}


int locktest(int nargs, char **args) {
	int i, result;

	(void)nargs;
	(void)args;

	inititems();
	kprintf("Starting lock test...\n");
```

```
	for (i=0; i<NTHREADS; i++) {
		result = thread_fork("synchtest", NULL, locktestthread,
				     NULL, i);
		if (result) {
			panic("locktest: thread_fork failed: %s\n",
			      strerror(result));
		}
	}
	for (i=0; i<NTHREADS; i++) {
		P(donesem);
	}

	kprintf("Lock test done.\n");

	return 0;
}

static void cvtestthread(void *junk, unsigned long num) {
	int i;
	volatile int j;
	struct timespec ts1, ts2;

	(void)junk;

	for (i=0; i<NCVLOOPS; i++) {
		lock_acquire(testlock);
		while (testval1 != num) {
			gettime(&ts1);
			cv_wait(testcv, testlock);
			gettime(&ts2);

			/* ts2 -= ts1 */
			timespec_sub(&ts2, &ts1, &ts2);

			/* Require at least 2000 cpu cycles (we're 25mhz) */
			if (ts2.tv_sec == 0 && ts2.tv_nsec < 40*2000) {
				kprintf("cv_wait took only %u ns\n",
					ts2.tv_nsec);
				kprintf("That's too fast... you must be "
					"busy-looping\n");
				V(donesem);
				thread_exit();
			}

		}
		kprintf("Thread %lu\n", num);
		testval1 = (testval1 + NTHREADS - 1)%NTHREADS;

		/*
		 * loop a little while to make sure we can measure the
		 * time waiting on the cv.
		 */
		for (j=0; j<3000; j++);

		cv_broadcast(testcv, testlock);
		lock_release(testlock);
	}
	V(donesem);
}

int cvtest(int nargs, char **args) {

	int i, result;
```

```
        (void)nargs;
        (void)args;

        inititems();
        kprintf("Starting CV test...\n");
        kprintf("Threads should print out in reverse order.\n");

        testval1 = NTHREADS-1;

        for (i=0; i<NTHREADS; i++) {
                result = thread_fork("synchtest", NULL, cvtestthread, NULL, i);
                if (result) {
                        panic("cvtest: thread_fork failed: %s\n",
                              strerror(result));
                }
        }
        for (i=0; i<NTHREADS; i++) {
                P(donesem);
        }

        kprintf("CV test done\n");

        return 0;
}

       ////////////////////////////////////////////////////////

#define NCVS 250
#define NLOOPS 40
static struct cv *testcvs[NCVS];
static struct lock *testlocks[NCVS];
static struct semaphore *gatesem;
static struct semaphore *exitsem;

static void sleepthread(void *junk1, unsigned long junk2) {
        unsigned i, j;

        (void)junk1;
        (void)junk2;

        for (j=0; j<NLOOPS; j++) {
                for (i=0; i<NCVS; i++) {
                        lock_acquire(testlocks[i]);
                        V(gatesem);
                        cv_wait(testcvs[i], testlocks[i]);
                        lock_release(testlocks[i]);
                }
                kprintf("sleepthread: %u\n", j);
        }
        V(exitsem);
       }

static void wakethread(void *junk1, unsigned long junk2) {
        unsigned i, j;

        (void)junk1;
        (void)junk2;

        for (j=0; j<NLOOPS; j++) {
                for (i=0; i<NCVS; i++) {
                        P(gatesem);
                        lock_acquire(testlocks[i]);
                        cv_signal(testcvs[i], testlocks[i]);
                        lock_release(testlocks[i]);
                }
```

```
                kprintf("wakethread: %u\n", j);
        }
        V(exitsem);
       }

       int cvtest2(int nargs, char **args) {
        unsigned i;
        int result;

        (void)nargs;
        (void)args;

        for (i=0; i<NCVS; i++) {
                testlocks[i] = lock_create("cvtest2 lock");
                testcvs[i] = cv_create("cvtest2 cv");
        }
        gatesem = sem_create("gatesem", 0);
        exitsem = sem_create("exitsem", 0);

        kprintf("cvtest2...\n");

        result = thread_fork("cvtest2", NULL, sleepthread, NULL, 0);
        if (result) {
                panic("cvtest2: thread_fork failed\n");
        }
        result = thread_fork("cvtest2", NULL, wakethread, NULL, 0);
        if (result) {
                panic("cvtest2: thread_fork failed\n");
        }

        P(exitsem);
        P(exitsem);

        sem_destroy(exitsem);
        sem_destroy(gatesem);
        exitsem = gatesem = NULL;
        for (i=0; i<NCVS; i++) {
                lock_destroy(testlocks[i]);
                cv_destroy(testcvs[i]);
                testlocks[i] = NULL;
                testcvs[i] = NULL;
        }

        kprintf("cvtest2 done\n");
        return 0;
}
```