**mutual exclusion 구현 학습**상호배제

컴퓨터공학과

2142851 김형준

운영체제 02분반

**목차**

1. **Dekker’s algorithm 구현**
   1. 소스 코드 및 설명
   2. 결과 화면
   3. 결과 분석
2. **Peterson’s algorithm 구현**
   1. 소스 코드 및 설명
   2. 결과 화면
   3. 결과 분석
3. **Dijkstra’s algorithm 구현**
   1. 소스 코드 및 설명
   2. 결과 화면
   3. 결과 분석
4. **Semaphore 구현**
   1. 소스 코드 및 설명
   2. 결과 화면
   3. 결과 분석
5. **알고리즘 종합 코드 및 시간 분석**
   1. 소스 코드
   2. 결과 화면
   3. 시간 분석

* **소스 코드 설명은 주석 처리를 통해 설명하였습니다.  
  (동일한 코드의 경우 중복하여 주석을 달지는 않았습니다.)**
* **목차 5번의 코드를 통해 모든 알고리즘을 한 코드에서 실행할 수 있도록 하였습니다.**
* **추가로, 5번 코드에 시간 측정 기능을 넣었습니다.**

1. **Dekker’s algorithm 구현**
2. 소스 코드 및 설명

*#include* <stdio.h>

*#include* <pthread.h>

int flag[2] = {0, 0}; *// 각 스레드의 플래그 상태를 저장하는 배열*

int turn = 0; *// 턴을 나타내는 변수*

int n[101]; *// 1부터 100까지의 숫자를 저장하는 배열*

int cnt = 0; *// 출력된 수의 카운트*

typedef struct {

    int start;

    int end;

    int thread\_id;

} args\_t;

void dekker\_critical\_section\_enter(int *thread\_id*) {

*// thread: 크리티컬 섹션 들어갈래*

    flag[thread\_id] = 1;

*// 상대 스레드의 flag가 올려져있다면*

*while* (flag[1 - thread\_id]) {

*// 내 차례도 아니라면*

*if* (turn != thread\_id) {

*// 대기..*

            flag[thread\_id] = 0;

*while* (turn != thread\_id) {}

*// 다시 들어가기*

            flag[thread\_id] = 1;

        }

    }

}

void dekker\_critical\_section\_exit(int *thread\_id*) {

*// 내 차례 끝났을 경우 turn 양도*

    turn = 1 - thread\_id;

*// 이제 안 들어갈거야.*

    flag[thread\_id] = 0;

}

void\* dekker\_func(void\* *args*) {

    args\_t\* thread\_args = (args\_t\*)args;

    int start = thread\_args->start;

    int end = thread\_args->end;

    int thread\_id = thread\_args->thread\_id;

*for* (int iter = start; iter <= end; iter++) {

*// 데커 알고리즘 크리티컬 영역 입장*

        dekker\_critical\_section\_enter(thread\_id);

        printf("Thread[%2d]: %3d \* 3 = %3d\n", thread\_id, n[iter], n[iter] \* 3);

        cnt++;

*// 데커 알고리즘 크리티컬 영역 퇴장*

        dekker\_critical\_section\_exit(thread\_id);

    }

    printf("=== thread[%2d] algorithm end ===\n", thread\_id);

*return* NULL;

}

int main() {

*for*(int i = 0; i < 101; i++)

        n[i] = i;

    pthread\_t threads[2];

    args\_t thread\_args[2];

*// thread 0번*

    thread\_args[0].start = 1;

    thread\_args[0].end = 50;

    thread\_args[0].thread\_id = 0;

*// thread 1번*

    thread\_args[1].start = 51;

    thread\_args[1].end = 100;

    thread\_args[1].thread\_id = 1;

*// create threads*

    pthread\_create(&threads[0], NULL, dekker\_func, &thread\_args[0]);

    pthread\_create(&threads[1], NULL, dekker\_func, &thread\_args[1]);

*// wait to end*

    pthread\_join(threads[0], NULL);

    pthread\_join(threads[1], NULL);

    printf("printf count: %d\n", cnt);

*return* 0;

}

1. 결과 화면

텍스트, 스크린샷, 폰트, 디자인이(가) 표시된 사진

자동 생성된 설명

1. 결과 분석
2. **Peterson’s algorithm 구현**
3. 소스 코드 및 설명

*#include* <stdio.h>

*#include* <pthread.h>

int flag[2] = {0, 0};

int turn = 0;

int n[101];

int cnt = 0;

typedef struct {

    int start;

    int end;

    int thread\_id;

} args\_t;

void peterson\_critical\_section\_enter(int *thread\_id*) {

*// thread 크리티컬 섹션 진입 시도*

    flag[thread\_id] = 1;

*// 현재 thread가 진입을 시도하므로, 상대 thread의 차례로 turn을 설정*

*// 너가 먼저 들어가*

    turn = 1 - thread\_id;

*// 상대 thread가 진입 시도 & 상대가 크리티컬 섹션에 진입할 차례인 경우 -> 대기*

*while* (flag[1 - thread\_id] && turn == 1 - thread\_id) {}

}

void peterson\_critical\_section\_exit(int *thread\_id*) {

    flag[thread\_id] = 0;

}

void\* peterson\_func(void\* *args*) {

    args\_t\* thread\_args = (args\_t\*)args;

    int start = thread\_args->start;

    int end = thread\_args->end;

    int thread\_id = thread\_args->thread\_id;

*for* (int iter = start; iter <= end; iter++) {

*// 피터슨 알고리즘 크리티컬 섹션 진입*

        peterson\_critical\_section\_enter(thread\_id);

        printf("Thread[%2d]: %3d \* 3 = %3d\n", thread\_id, n[iter], n[iter] \* 3);

        cnt++;

*// 피터슨 알고리즘 크리티컬 섹션 퇴장*

        peterson\_critical\_section\_exit(thread\_id);

    }

    printf("=== thread[%2d] algorithm end ===\n", thread\_id);

*return* NULL;

}

int main() {

*for*(int i = 0; i < 101; i++)

        n[i] = i;

    pthread\_t threads[2];

    args\_t thread\_args[2];

*// Thread 0번*

    thread\_args[0].start = 1;

    thread\_args[0].end = 50;

    thread\_args[0].thread\_id = 0;

*// Thread 1번*

    thread\_args[1].start = 51;

    thread\_args[1].end = 100;

    thread\_args[1].thread\_id = 1;

*// thread create*

    pthread\_create(&threads[0], NULL, peterson\_func, &thread\_args[0]);

    pthread\_create(&threads[1], NULL, peterson\_func, &thread\_args[1]);

*// wait to end*

    pthread\_join(threads[0], NULL);

    pthread\_join(threads[1], NULL);

    printf("printf count: %d\n", cnt);

*return* 0;

}

1. 결과 화면

텍스트, 스크린샷, 폰트, 디자인이(가) 표시된 사진

자동 생성된 설명

1. 결과 분석
2. **Dijkstra’s algorithm 구현**
3. 소스 코드 및 설명

*#include* <stdio.h>

*#include* <pthread.h>

*#define* N 4 *// thread count*

*#define* RANGE 100

*// thread flag*

enum state { IDLE = 0, WANT\_IN, IN\_CS };

int flag[N] = {IDLE};

int turn = 0;

int n[101];

int cnt = 0;

typedef struct {

    int start;

    int end;

    int thread\_id;

} args\_t;

void dijkstra\_critical\_section\_enter(int *thread\_id*) {

    int j;

*do* {

*// 임계 영역 진입 시도 1단계*

        flag[thread\_id] = WANT\_IN;

*while* (turn != thread\_id) {

*if* (flag[turn] == IDLE) {

                turn = thread\_id;

            }

        }

*// 임계 영역 진입 시도 2단계*

        flag[thread\_id] = IN\_CS;

        j = 0;

*// 자신 이외에 in-CS 영역이 있는지 확인*

*while* (j < N && (j == thread\_id || flag[j] != IN\_CS)) {

            j++;

        }

    } *while* (j < N);

*// 없으면 내가 크리티컬 섹션에 들어가기*

}

void dijkstra\_critical\_section\_exit(int *thread\_id*) {

*// flag를 IDLE로 설정*

    flag[thread\_id] = IDLE;

*// 다음 턴으로*

    turn = (turn + 1) % N;

}

void\* dijkstra\_func(void\* *args*) {

    args\_t\* thread\_args = (args\_t\*)args;

    int start = thread\_args->start;

    int end = thread\_args->end;

    int thread\_id = thread\_args->thread\_id;

*for* (int iter = start; iter <= end; iter++) {

        dijkstra\_critical\_section\_enter(thread\_id); *// Dijkstra 임계 영역 진입*

        printf("Thread[%d]: %3d \* 3 = %3d\n", thread\_id, n[iter], n[iter] \* 3);

        cnt++;

        dijkstra\_critical\_section\_exit(thread\_id); *// Dijkstra 임계 영역 종료*

    }

    printf("=== thread[%2d] algorithm end ===\n", thread\_id);

*return* NULL;

}

int main() {

*// 숫자 배열 초기화*

*for* (int i = 0; i < 101; i++) {

        n[i] = i;

    }

    pthread\_t threads[N];

    args\_t thread\_args[N];

*// 각 스레드의 범위 설정*

    int step = RANGE / N;

*for* (int i = 0; i < N; i++) {

        thread\_args[i].start = i \* step + 1;

        thread\_args[i].end = (i + 1) \* step;

        thread\_args[i].thread\_id = i;

*// 스레드 생성*

        pthread\_create(&threads[i], NULL, dijkstra\_func, &thread\_args[i]);

    }

*// wait to end*

*for* (int i = 0; i < N; i++) {

        pthread\_join(threads[i], NULL);

    }

*// 최종 출력 카운트*

    printf("printf count: %d\n", cnt);

*return* 0;

}

1. 결과 화면

텍스트, 스크린샷, 디자인이(가) 표시된 사진

자동 생성된 설명텍스트, 스크린샷, 폰트, 디자인이(가) 표시된 사진

자동 생성된 설명

1. 결과 분석
2. **Semaphore 구현**
3. 소스 코드 및 설명

*#include* <stdio.h>

*#include* <pthread.h>

*#include* <semaphore.h>

*#define* RANGE 100

*#define* THREAD\_COUNT 4

*// 세마포 객체*

sem\_t sem;

int n[101];

int cnt = 0;

typedef struct {

    int start;

    int end;

    int thread\_id;

} args\_t;

void\* semaphore\_func(void\* *args*) {

    args\_t\* thread\_args = (args\_t\*)args;

    int start = thread\_args->start;

    int end = thread\_args->end;

    int thread\_id = thread\_args->thread\_id;

*for* (int iter = start; iter <= end; iter++) {

*// 세마포를 통해 크리티컬 섹션 진입 제어*

        sem\_wait(&sem);

        printf("Thread[%d]: %3d \* 3 = %3d\n", thread\_id, n[iter], n[iter] \* 3);

        cnt++;

*// 크리티컬 섹션 종료 후 세마포 인카운트*

        sem\_post(&sem);

    }

    printf("=== thread[%2d] algorithm end ===\n", thread\_id);

*return* NULL;

}

int main() {

*for* (int i = 0; i < 101; i++) {

        n[i] = i;

    }

*// 세마포 초기값*

    sem\_init(&sem, 0, 1);

    pthread\_t threads[THREAD\_COUNT];

    args\_t thread\_args[THREAD\_COUNT];

*// thread 범위*

    int step = RANGE / THREAD\_COUNT;

*for* (int i = 0; i < THREAD\_COUNT; i++) {

        thread\_args[i].start = i \* step + 1;

        thread\_args[i].end = (i + 1) \* step;

        thread\_args[i].thread\_id = i;

*// thread create*

        pthread\_create(&threads[i], NULL, semaphore\_func, &thread\_args[i]);

    }

*// wait to end*

*for* (int i = 0; i < THREAD\_COUNT; i++) {

        pthread\_join(threads[i], NULL);

    }

*// 세마포 파괴!!*

    sem\_destroy(&sem);

    printf("Total count: %d\n", cnt);

*return* 0;

}

1. 결과 화면

텍스트, 스크린샷, 폰트, 디자인이(가) 표시된 사진

자동 생성된 설명

1. 결과 분석
2. **알고리즘 종합 코드 및 시간 분석**
3. 소스 코드

*#include* <stdio.h>

*#include* <pthread.h>

*#include* <semaphore.h>

*#include* <stdlib.h>

*#define* RANGE 100 *// 숫자 범위*

int turn = 0;

int n[101];

int cnt = 0;

int\* flag;

int thread\_count;

typedef struct {

    int start;

    int end;

    int thread\_id;

} args\_t;

*// 1. Dekker*

void dekker\_critical\_section\_enter(int *thread\_id*);

void dekker\_critical\_section\_exit(int *thread\_id*);

void\* dekker\_func(void\* *args*);

*// 2. Peterson*

void peterson\_critical\_section\_enter(int *thread\_id*);

void peterson\_critical\_section\_exit(int *thread\_id*);

void\* peterson\_func(void\* *args*);

*// 3. Dijkstra*

void dijkstra\_critical\_section\_enter(int *thread\_id*);

void dijkstra\_critical\_section\_exit(int *thread\_id*);

void\* dijkstra\_func(void\* *args*);

enum state { IDLE = 0, WANT\_IN, IN\_CS };

*// 4. Semaphore*

void\* semaphore\_func(void\* *args*);

sem\_t sem;

char algorithms[5][30] = { "exit", "Dekker's Algorithm", "Peterson's Algorithm", "Dijkstra's Algorithm", "Semaphore" };

int main() {

*for* (int i = 0; i< 101; i++) {

        n[i] = i;

    }

*while*(1) {

        int input;

        cnt = 0;

        turn = 0;

        printf("<Select Your Algorithm>\n");

        printf("---------- Select ----------\n");

*for*(int i = 0; i < 5; i++) {

            printf("%2d. %s\n", i, algorithms[i]);

        }

        printf("\n");

        printf("input: ");

*if* (scanf("%d", &input) != 1) {

*while* (getchar() != '\n'); *// 입력 스트림을 비움*

            system("clear");

            printf("Invalid input. (input type: int)\n\n");

*continue*; *// 다시 입력 받기*

        }

*if* (input == 0) *break*;

*else* *if* (input == 1 || input == 2 || input == 3 || input == 4) {

            printf("================ %s Start ================\n", algorithms[input]);

        }

*else* {

            system("clear");

            printf("Wrong input value\n\n");

*continue*;

        }

        pthread\_t\* threads;

        args\_t\* thread\_args;

        int thread\_args\_count;

*if* (input == 1 || input == 2) {

            thread\_count = 2;

        }

*else* *if* (input == 3 || input == 4) {

            thread\_count = 4;

        }

*// semaphore first setting*

*if* (input == 4) {

            sem\_init(&sem, 0, 1);

        }

*// thread malloc*

        threads = (pthread\_t\*)malloc(sizeof(pthread\_t) \* thread\_count);

        thread\_args = (args\_t\*)malloc(sizeof(args\_t) \* thread\_count);

*// flag malloc*

        flag = (int\*)malloc(sizeof(int) \* thread\_count);

*// flag setting*

*if* (input == 1 || input == 2 || input == 3) {

*for* (int i = 0; i < thread\_count; i++) {

                flag[i] = 0;

            }

        }

*else* *if* (input == 3) {

*for* (int i = 0; i < thread\_count; i++) {

                flag[i] = IDLE;

            }

        }

        int step = RANGE / thread\_count;

        double aveTime = 0;

*for* (int repeat = 0; repeat < 10; repeat++) {

*// TIME!!*

            clock\_t startTime, endTime;

            double totalTime;

            startTime = clock();

*// thread setting*

*for* (int i = 0; i < thread\_count; i++) {

                thread\_args[i].start = i \* step + 1;

                thread\_args[i].end = (i + 1) \* step;

                thread\_args[i].thread\_id = i;

*// thread create*

*if* (input == 1) pthread\_create(&threads[i], NULL, dekker\_func, &thread\_args[i]);

*else* *if* (input == 2) pthread\_create(&threads[i], NULL, peterson\_func, &thread\_args[i]);

*else* *if* (input == 3) pthread\_create(&threads[i], NULL, dijkstra\_func, &thread\_args[i]);

*else* *if* (input == 4) pthread\_create(&threads[i], NULL, semaphore\_func, &thread\_args[i]);

            }

*/\* function \*/*

*// wait to the end*

*for* (int i = 0; i < thread\_count; i++) {

                pthread\_join(threads[i], NULL);

            }

*// destroy semaphore*

*if* (input == 4) {

                sem\_destroy(&sem);

            }

*// TIME!!*

            endTime = clock();

            totalTime = (double)(endTime - startTime) / CLOCKS\_PER\_SEC;

            printf("\n");

            printf("================ Result ================\n");

            printf("Algorithm           | %s\n", algorithms[input]);

            printf("Mutual exclusion    | %s (cnt: %d)\n", ((cnt == 100) ? "succeeded!" : "failed."), cnt);

            printf("Thread count        | %d\n", thread\_count);

            printf("Total Time          | %f seconds\n", totalTime);

            printf("================= End =================\n");

            printf("\n");

            aveTime += totalTime;

        }

        printf("[%lf]\n", aveTime/10);

        free(flag);

        free(threads);

        free(thread\_args);

*// system("pause");*

        printf("Press Enter to continue :)");

        getchar(); *// Wait to enter 'Enter' :)*

        getchar();

        system("clear");

    }

*return* 0;

}

*/\* 1. Dekker \*/*

void dekker\_critical\_section\_enter(int *thread\_id*) {

*// thread: 크리티컬 섹션 들어갈래*

    flag[thread\_id] = 1;

*// 상대 스레드의 flag가 올려져있다면*

*while* (flag[1 - thread\_id]) {

*// 내 차례도 아니라면*

*if* (turn != thread\_id) {

*// 대기..*

            flag[thread\_id] = 0;

*while* (turn != thread\_id) {}

*// 다시 들어가기*

            flag[thread\_id] = 1;

        }

    }

}

void dekker\_critical\_section\_exit(int *thread\_id*) {

*// 내 차례 끝났을 경우 turn 양도*

    turn = 1 - thread\_id;

*// 이제 안 들어갈거야.*

    flag[thread\_id] = 0;

}

void\* dekker\_func(void\* *args*) {

    args\_t\* thread\_args = (args\_t\*)args;

    int start = thread\_args->start;

    int end = thread\_args->end;

    int thread\_id = thread\_args->thread\_id;

*for* (int iter = start; iter <= end; iter++) {

*// 데커 알고리즘 크리티컬 영역 입장*

        dekker\_critical\_section\_enter(thread\_id);

        printf("Thread[%2d]: %3d \* 3 = %3d\n", thread\_id, n[iter], n[iter] \* 3);

        cnt++;

*// 데커 알고리즘 크리티컬 영역 퇴장*

        dekker\_critical\_section\_exit(thread\_id);

    }

    printf("=== thread[%2d] algorithm end ===\n", thread\_id);

*return* NULL;

}

*/\* 2. Peterson \*/*

void peterson\_critical\_section\_enter(int *thread\_id*) {

*// thread 크리티컬 섹션 진입 시도*

    flag[thread\_id] = 1;

*// 현재 thread가 진입을 시도하므로, 상대 thread의 차례로 turn을 설정*

*// 너가 먼저 들어가*

    turn = 1 - thread\_id;

*// 상대 thread가 진입 시도 & 상대가 크리티컬 섹션에 진입할 차례인 경우 -> 대기*

*while* (flag[1 - thread\_id] && turn == 1 - thread\_id) {}

}

void peterson\_critical\_section\_exit(int *thread\_id*) {

    flag[thread\_id] = 0;

}

void\* peterson\_func(void\* *args*) {

    args\_t\* thread\_args = (args\_t\*)args;

    int start = thread\_args->start;

    int end = thread\_args->end;

    int thread\_id = thread\_args->thread\_id;

*for* (int iter = start; iter <= end; iter++) {

*// 피터슨 알고리즘 크리티컬 섹션 진입*

        peterson\_critical\_section\_enter(thread\_id);

        printf("Thread[%2d]: %3d \* 3 = %3d\n", thread\_id, n[iter], n[iter] \* 3);

        cnt++;

*// 피터슨 알고리즘 크리티컬 섹션 퇴장*

        peterson\_critical\_section\_exit(thread\_id);

    }

    printf("=== thread[%2d] algorithm end ===\n", thread\_id);

*return* NULL;

}

*/\* 3. Dijkstra \*/*

void dijkstra\_critical\_section\_enter(int *thread\_id*) {

    int j;

*do* {

*// 임계 영역 진입 시도 1단계*

        flag[thread\_id] = WANT\_IN;

*while* (turn != thread\_id) {

*if* (flag[turn] == IDLE) {

                turn = thread\_id;

            }

        }

*// 임계 영역 진입 시도 2단계*

        flag[thread\_id] = IN\_CS;

        j = 0;

*// 자신 이외에 in-CS 영역이 있는지 확인*

*while* (j < thread\_count && (j == thread\_id || flag[j] != IN\_CS)) {

            j++;

        }

    } *while* (j < thread\_count);

*// 없으면 내가 크리티컬 섹션에 들어가기*

}

void dijkstra\_critical\_section\_exit(int *thread\_id*) {

*// flag를 IDLE로 설정*

    flag[thread\_id] = IDLE;

*// 다음 턴으로*

    turn = (turn + 1) % thread\_count;

}

void\* dijkstra\_func(void\* *args*) {

    args\_t\* thread\_args = (args\_t\*)args;

    int start = thread\_args->start;

    int end = thread\_args->end;

    int thread\_id = thread\_args->thread\_id;

*for* (int iter = start; iter <= end; iter++) {

        dijkstra\_critical\_section\_enter(thread\_id); *// Dijkstra 임계 영역 진입*

        printf("Thread[%d]: %3d \* 3 = %3d\n", thread\_id, n[iter], n[iter] \* 3);

        cnt++;

        dijkstra\_critical\_section\_exit(thread\_id); *// Dijkstra 임계 영역 종료*

    }

    printf("=== thread[%2d] algorithm end ===\n", thread\_id);

*return* NULL;

}

*/\* 4. Semaphore \*/*

void\* semaphore\_func(void\* *args*) {

    args\_t\* thread\_args = (args\_t\*)args;

    int start = thread\_args->start;

    int end = thread\_args->end;

    int thread\_id = thread\_args->thread\_id;

*for* (int iter = start; iter <= end; iter++) {

*// 세마포를 통해 크리티컬 섹션 진입 제어*

        sem\_wait(&sem);

        printf("Thread[%d]: %3d \* 3 = %3d\n", thread\_id, n[iter], n[iter] \* 3);

        cnt++;

*// 크리티컬 섹션 종료 후 세마포 인카운트*

        sem\_post(&sem);

    }

    printf("=== thread[%2d] algorithm end ===\n", thread\_id);

*return* NULL;

}

1. 결과 화면

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

input: 3인 경우 (다익스트라 알고리즘)

텍스트, 스크린샷, 폰트, 디자인이(가) 표시된 사진

자동 생성된 설명

1. 시간 분석

\* 시간 측정 및 분석을 진행하였으나, 실행 간 편차가 커서 정확한 측정이 불가했습니다.

- Dekker’s Algorithm

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명 텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

- Peterson’s Algorithm

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

- Dijkstra’s Algorithm

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

- Semaphore

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

1회차 평균 (10회)

|  |  |  |  |
| --- | --- | --- | --- |
| Dekker | Peterson | Dijkstra | Semaphore |
| 0.000304 | 0.000264 | 0.001456 | 0.000583 |

2회차 평균 (10회)

|  |  |  |  |
| --- | --- | --- | --- |
| Dekker | Peterson | Dijkstra | Semaphore |
| 0.000510 | 0.000242 | 0.079997 | 0.000622 |