**multi-process, multi-thread  
구현 학습**운영체제 02분반

컴퓨터공학과

2142851 김형준

**목차**

1. **소스 코드**
   1. 멀티 프로세스, 멀티 스레드
   2. 싱글 프로세스
   3. 코드 설명 (mul-pro-th.c)
   4. 출력 결과
2. **결과 분석**
   1. multi코드와 single코드의 시간 측정
   2. 추가 실험
3. **문제점과 해결 방법**
   1. 병렬 처리 문제
   2. thread 전달 인자 구체화
   3. console 가시성 증가
   4. 몇 프로세스가 실행되지 않는 문제
   5. start, end 값 문제
4. **소스 코드**

(보고서의 ‘3. 문제점과 해결 방법’을 거친 최종 코드입니다.)

1. **멀티 프로세스, 멀티 쓰레드**

mul-pro-th.c

*#include* <stdio.h>

*#include* <stdlib.h>

*#include* <sys/types.h>

*#include* <sys/wait.h>

*#include* <pthread.h>

*#include* <unistd.h>

*#include* <time.h>

*#define* RANGE 100

*#define* THREADCOUNT 4

*#define* PROCESSCOUNT 4

*#define* REPEAT 10

*// int testCnt;*

void\* func(void\* *args*) {

    int\* params = (int\*)args;

    int mul\_value = params[0];

    int start = params[1];

    int end = params[2];

    int process\_num = params[3];

    int thread\_num = params[4];

*// // TEST*

*// printf("\n");*

*// printf("mul\_value: %d\n", mul\_value);*

*// printf("start: %d\n", start);*

*// printf("end: %d\n", end);*

*// printf("process\_num: %d\n", process\_num);*

*// printf("thread\_num: %d\n", thread\_num);*

*// printf("\n");*

    printf("[Process[%d]-thread[%d] start]\n", process\_num, thread\_num);

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

        printf("%3d \* %3d = %3d\n", i, mul\_value, i \* mul\_value);

    }

    printf("[Process[%d]-thread[%d] end]\n", process\_num, thread\_num);

*// args 배열 해제*

    free(args);

    pthread\_exit(NULL);

}

*// process task*

void process\_task(int *process\_num*) {

    pthread\_t tid[THREADCOUNT];

    printf("[Process %d start]\n", process\_num);

*/// thread 생성*

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

*// 각 스레드마다 args 배열을 동적으로 할당 + 초기화*

        int\* args = (int \*)malloc(5 \* sizeof(int));

*if* (args == NULL) {

            fprintf(stderr, "Memory allocation failed!!\n");

            exit(1);

        }

*// int args[5];*

        args[0] = process\_num \* 2 + 1; *// 곱할 값 [3, 5, 7, 9]*

        args[2] = (i + 1) \* RANGE / THREADCOUNT; *// 종료 값 [25, 50, 75, 100]*

        args[1] = args[2] - RANGE / THREADCOUNT + 1; *// 시작 값 [1, 26, 51, 76]*

        args[3] = process\_num; *// process num*

        args[4] = i + 1; *// thread num*

*if* (pthread\_create(&tid[i], NULL, func, args) != 0) {

            fprintf(stderr, "thread create error\n");

            exit(1);

        }

*// TEST*

*// testCnt++;*

*// printf("========= i: %d, mul: %d, process num: %d, thread num: %d, start: %d, end: %d]\n", i, args[0], args[3], args[4], args[1], args[2]);*

    }

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

        pthread\_join(tid[i], NULL);

    }

    printf("[Process %d end]\n", process\_num);

}

int main() {

*// 반복 후 평균을 내기 위해 REPEAT회 결과값을 더하고 저장할 변수*

    double totalSum = 0;

*// 반복 횟수*

*// REPEAT만큼 반복*

*for* (int testCase = 0; testCase < REPEAT; testCase++) {

*// testCnt = 0;*

*// startTime: 시작 시간*

*// endTime: 끝 시간*

        clock\_t startTime, endTime;

*// totalTime: startTime과 endTime을 연산 후 나온 결과 (실행 시간)*

        double totalTime;

        startTime = clock(); *// 시작 시간 기록*

*// pid: 프로세스 ID 저장*

        pid\_t pid;

*// 프로세스의 개수만큼 반복*

*for* (int i = 1; i <= PROCESSCOUNT; i++) {

*// 자식 프로세스 생성*

            pid = fork();

*if* (pid < 0) { *// fork 실패*

                perror("fork");

                exit(1);

            }

*else* *if* (pid == 0) { *// 자식 프로세스*

*// process\_task 함수로 이동*

*// process\_task(시작 값, 마지막 값, 프로세스 번호)*

                process\_task(i);

                exit(0);

            }

        }

*// 자식 프로세스 종료 대기*

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

            wait(NULL);

        }

        endTime = clock(); *// 종료 시간*

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

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

        totalSum += totalTime;

    }

*// 평균 시간 계산 후 출력*

    printf("[========= mul-pro-th.c END =========]\n");

    printf("[Average time after %d repetitions: %f seconds\n", REPEAT, totalSum / REPEAT);

*// printf("testCnt: %d\n", testCnt);*

*return* 0;

}

1. **싱글 프로세스**

sin-pro.c

*#include* <stdio.h>

*#include* <time.h>

*#define* REPEAT 10

double totalSum = 0;

void task() {

    clock\_t startTime, endTime;

    double totalTime;

    startTime = clock();

*for* (int i = 1; i <= 4; i++) {

        printf("[%2d]\n", i);

*for* (int j = 1; j <= 100; j++) {

            printf("%3d \* %3d = %3d\n", j, i \* 2 + 1, j \* (i \* 2 + 1));

        }

    }

    endTime = clock();

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

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

    totalSum += totalTime;

}

int main() {

*for* (int testCase = 0; testCase < REPEAT; testCase++) {

        task();

    }

    printf("[========= sin-pro.c END =========]\n");

    printf("[Average time after %d repetitions: %f seconds]\n", REPEAT, totalSum / REPEAT);

*return* 0;

}

1. **코드 설명 (mul-pro-th.c)**

<#define>

*#define* RANGE 100

*#define* THREADCOUNT 4

*#define* PROCESSCOUNT 4

*#define* REPEAT 10

RANGE: 1~n까지의 숫자를 연산

THREADCOUNT: Thread의 개수

PROCESSCOUNT: Process의 개수

REPEAT: 평균 도출을 위한 반복 횟수

<func>

void\* func(void\* *args*) {

    int\* params = (int\*)args;

    int mul\_value = params[0];

    int start = params[1];

    int end = params[2];

    int process\_num = params[3];

    int thread\_num = params[4];

*// // TEST*

*// printf("\n");*

*// printf("mul\_value: %d\n", mul\_value);*

*// printf("start: %d\n", start);*

*// printf("end: %d\n", end);*

*// printf("process\_num: %d\n", process\_num);*

*// printf("thread\_num: %d\n", thread\_num);*

*// printf("\n");*

    printf("[Process[%d]-thread[%d] start]\n", process\_num, thread\_num);

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

        printf("%3d \* %3d = %3d\n", i, mul\_value, i \* mul\_value);

    }

    printf("[Process[%d]-thread[%d] end]\n", process\_num, thread\_num);

*// args 배열 해제*

    free(args);

    pthread\_exit(NULL);

}

process 생성 시 process의 번호, 곱할 인자(3, 5, 7, 9), 시작, 끝, 프로세스 번호, 스레드 번호를 출력 및 연산할 수 있도록 변수를 만들고, args의 값에서 변수값을 저장했습니다.

스레드의 시작을 알리는 printf를 출력합니다.

for문을 통해 연산, 출력을 진행합니다.

이후 스레드의 끝을 알리는 printf를 출력하고 args 변수를 초기화한 후에 pthread\_exit 함수를 실행합니다.

<process\_task>

*// process task*

void process\_task(int *process\_num*) {

    pthread\_t tid[THREADCOUNT];

    printf("[Process %d start]\n", process\_num);

*/// thread 생성*

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

*// 각 스레드마다 args 배열을 동적으로 할당 + 초기화*

        int\* args = (int \*)malloc(5 \* sizeof(int));

*if* (args == NULL) {

            fprintf(stderr, "Memory allocation failed!!\n");

            exit(1);

        }

*// int args[5];*

        args[0] = process\_num \* 2 + 1; *// 곱할 값 [3, 5, 7, 9]*

        args[2] = (i + 1) \* RANGE / THREADCOUNT; *// 종료 값 [25, 50, 75, 100]*

        args[1] = args[2] - RANGE / THREADCOUNT + 1; *// 시작 값 [1, 26, 51, 76]*

        args[3] = process\_num; *// process num*

        args[4] = i + 1; *// thread num*

*if* (pthread\_create(&tid[i], NULL, func, args) != 0) {

            fprintf(stderr, "thread create error\n");

            exit(1);

        }

*// TEST*

*// testCnt++;*

*// printf("========= i: %d, mul: %d, process num: %d, thread num: %d, start: %d, end: %d]\n", i, args[0], args[3], args[4], args[1], args[2]);*

    }

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

        pthread\_join(tid[i], NULL);

    }

    printf("[Process %d end]\n", process\_num);

}

process의 넘버를 인자로 받는 함수입니다.

THREADCOUNT만큼의 pthread\_t 타입 tid 변수를 생성합니다.

프로세스의 시작을 알리는 printf를 출력합니다.

for문을 THREADCOUNT의 값만큼 반복합니다.

args 변수를 동적으로 할당하고, 할당 실패 시 메시지를 출력합니다.

args 변수에 값을 저장합니다.

|  |  |
| --- | --- |
| args[0] | 곱할 값 (3, 5, 7, 9) |
| args[1] | 시작 값 (1, 26, 51, 76) |
| args[2] | 종료 값 (25, 50, 75, 100) |
| args[3] | Process Number |
| args[4] | Thread Number |

pthread\_create 함수를 통해 스레드를 생성합니다.

|  |  |
| --- | --- |
| &tid[i] | 생성된 스레드의 식별자를 받을 포인터 |
| NULL | 스레드의 속성 (기본적으로 NULL 사용) |
| func | 스레드가 실행할 함수의 포인터 |
| args | func 함수에 전달할 인수 |

for문을 통해 생성된 각 스레드가 종료될 때까지 기다립니다.

종료되었을 경우, 스레드의 리소스를 해제하고, 종료 상태를 확인합니다.

<main>

int main() {

*// 반복 후 평균을 내기 위해 REPEAT회 결과값을 더하고 저장할 변수*

    double totalSum = 0;

*// 반복 횟수*

*// REPEAT만큼 반복*

*for* (int testCase = 0; testCase < REPEAT; testCase++) {

*// testCnt = 0;*

*// startTime: 시작 시간*

*// endTime: 끝 시간*

        clock\_t startTime, endTime;

*// totalTime: startTime과 endTime을 연산 후 나온 결과 (실행 시간)*

        double totalTime;

        startTime = clock(); *// 시작 시간 기록*

*// pid: 프로세스 ID 저장*

        pid\_t pid;

*// 프로세스의 개수만큼 반복*

*for* (int i = 1; i <= PROCESSCOUNT; i++) {

*// 자식 프로세스 생성*

            pid = fork();

*if* (pid < 0) { *// fork 실패*

                perror("fork");

                exit(1);

            }

*else* *if* (pid == 0) { *// 자식 프로세스*

*// process\_task 함수로 이동*

*// process\_task(시작 값, 마지막 값, 프로세스 번호)*

                process\_task(i);

                exit(0);

            }

        }

*// 자식 프로세스 종료 대기*

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

            wait(NULL);

        }

        endTime = clock(); *// 종료 시간*

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

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

        totalSum += totalTime;

    }

*// 평균 시간 계산 후 출력*

    printf("[========= mul-pro-th.c END =========]\n");

    printf("[Average time after %d repetitions: %f seconds\n", REPEAT, totalSum / REPEAT);

*// printf("testCnt: %d\n", testCnt);*

*return* 0;

}

main 함수의 경우 이전 보고서의 해석과 동일합니다.

주석을 통해 분석 결과를 작성하였습니다.

1. **출력 결과**

시작 부분

텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

끝 부분

텍스트, 스크린샷, 디자인이(가) 표시된 사진

자동 생성된 설명

1. **결과 분석**

(단위: second)

1. **multi코드와 single코드의 시간 측정**

<1회 실행 시간>

1회차





|  |  |
| --- | --- |
| **sin-pro.c (싱글)** | **mul-pro-th.c (멀티 프로세스, 쓰레드)** |
| 0.000462 | 0.000417 |

2회차

텍스트, 폰트, 스크린샷, 라인이(가) 표시된 사진

자동 생성된 설명



|  |  |
| --- | --- |
| **sin-pro.c (싱글)** | **mul-pro-th.c (멀티 프로세스, 쓰레드)** |
| 0.001044 | 0.000394 |

1. **추가 실험**

1회 실험으로는 실행 시간 마다 편차가 크다고 판단하여, 여러 번 실행 후 평균을 구하기로 하였습니다.

<10회 반복 후 평균 시간>

1회차

텍스트, 폰트, 스크린샷, 라인이(가) 표시된 사진

자동 생성된 설명



|  |  |
| --- | --- |
| **sin-pro.c (싱글)** | **mul-pro-th.c (멀티 프로세스, 쓰레드)** |
| 0.000793 | 0.000301 |

2회차

텍스트, 폰트, 스크린샷, 라인이(가) 표시된 사진

자동 생성된 설명

텍스트, 폰트, 스크린샷, 라인이(가) 표시된 사진

자동 생성된 설명

|  |  |
| --- | --- |
| **sin-pro.c (싱글)** | **mul-pro-th.c (멀티 프로세스, 쓰레드)** |
| 0.001198 | 0.000329 |

<1,000회 반복 후 평균 시간>





|  |  |
| --- | --- |
| **sin-pro.c (싱글)** | **mul-pro-th.c (멀티 프로세스, 쓰레드)** |
| 0.000722 | 0.000306 |

1. **문제점과 해결 방법**

(코드의 경우 변경 부분을 형관펜으로 표시하였습니다.)

1. **병렬 처리 문제**

함수 process\_task 부분에서 병렬 처리가 안되는 문제가 있었습니다.

*// process task*

void process\_task(int *process\_num*) {

    pthread\_t tid[THREADCOUNT];

    printf("[Process %d start]\n", process\_num);

*// thread 생성*

*for* (int i = 1; i <= THREADCOUNT; i++) {

        int args[3];

        args[0] = process\_num \* 2 + 1; *// 곱할 값 [3, 5, 7, 9]*

        args[2] = i \* RANGE / THREADCOUNT; *// 종료 값 [25, 50, 75, 100]*

        args[1] = args[2] - RANGE / THREADCOUNT + 1; *// 시작 값 [1, 26, 51, 76]*

*if* (pthread\_create(&tid[i], NULL, func, &args) != 0) {

            fprintf(stderr, "thread create error\n");

            exit(1);

        }

        pthread\_join(tid[i], NULL);

    }

    printf("[Process %d end]\n", process\_num);

}

(원래 코드)

pthread\_join 함수의 경우 쓰레드를 실행한 후, 쓰레드가 종료될 때까지 기다립니다.

성공 시 0을 반환하고, 문제가 있는 경우, 에러를 반환합니다.

<문제되는 순서 (쓰레드 2개로 가정)>

쓰레드1 생성 – 쓰레드1 종료 대기 – 쓰레드2 생성 – 쓰레드2 종료 대기

<해결된 순서>

쓰레드1 생성 – 쓰레드 2 생성 – 쓰레드 1 종료 대기 – 쓰레드 2 종료 대기

<해결된 코드>

*// process task*

void process\_task(int *process\_num*) {

    pthread\_t tid[THREADCOUNT];

    printf("[Process %d start]\n", process\_num);

*// thread 생성*

*for* (int i = 1; i <= THREADCOUNT; i++) {

        int args[3];

        args[0] = process\_num \* 2 + 1; *// 곱할 값 [3, 5, 7, 9]*

        args[2] = i \* RANGE / THREADCOUNT; *// 종료 값 [25, 50, 75, 100]*

        args[1] = args[2] - RANGE / THREADCOUNT + 1; *// 시작 값 [1, 26, 51, 76]*

*if* (pthread\_create(&tid[i], NULL, func, &args) != 0) {

            fprintf(stderr, "thread create error\n");

            exit(1);

        }

    }

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

        pthread\_join(tid[i], NULL);

    }

    printf("[Process %d end]\n", process\_num);

}

<변경 전후 시간 측정>

변경 전



변경 후

텍스트, 폰트, 스크린샷, 그래픽이(가) 표시된 사진

자동 생성된 설명

|  |  |
| --- | --- |
| **변경 전 (순차 처리)** | **변경 후 (병렬 처리)** |
| 0.000393 seconds | 0.000291 seconds |

1. **thread 전달 인자 구체화**

thread에 전달할 인자를 동적 할당을 이용하여 전달해보고 싶었습니다.

따라서 코드에 다음 부분을 추가하였습니다.

typedef struct {

    int mul\_value;

    int start;

    int end;

} ThreadArgs;

void\* func(void\* *args*) {

    ThreadArgs\* threadArgs = (ThreadArgs\*)args;

    printf("[thread start]\n");

*for* (int i = threadArgs->start; i <= threadArgs->end; i++) {

        printf("%3d \* %3d = %3d\n", i, threadArgs->mul\_value, i \* threadArgs->mul\_value);

    }

    printf("[thread end]\n");

    pthread\_exit(NULL);

}

*// process task*

void process\_task(int *process\_num*) {

    pthread\_t tid[THREADCOUNT];

    printf("[Process %d start]\n", process\_num);

*// thread 생성*

*for* (int i = 1; i <= THREADCOUNT; i++) {

        ThreadArgs\* args = (ThreadArgs\*)malloc(sizeof(ThreadArgs));

*if* (args == NULL) {

            fprintf(stderr, "memory allocation error\n");

            exit(1);

        }

        args->mul\_value = process\_num \* 2 + 1; *// 곱할 값 [3, 5, 7, 9]*

        args->end = i \* RANGE / THREADCOUNT; *// 종료 값 [25, 50, 75, 100]*

        args->start = args->end - RANGE / THREADCOUNT + 1; *// 시작 값 [1, 26, 51, 76]*

*if* (pthread\_create(&tid[i], NULL, func, &args) != 0) {

            fprintf(stderr, "thread create error\n");

            exit(1);

        }

    }

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

        pthread\_join(tid[i], NULL);

    }

    printf("[Process %d end]\n", process\_num);

}

하지만 위와 같은 방법으로 실행 시 동적 메모리 할당에 시간을 소비하는 부분이 생길 것이라고 생각하였고, 이러한 문제 때문에, 정적으로 할당하는 방식을 채택하였습니다.

또한, 해당 방식(코드)의 동적 할당 부분에 있어, 오류가 발생하는 것으로 보여, 사용하지 않았습니다.

int args[3];

args[0] = process\_num \* 2 + 1; *// 곱할 값 [3, 5, 7, 9]*

args[2] = i \* RANGE / THREADCOUNT; *// 종료 값 [25, 50, 75, 100]*

args[1] = args[2] - RANGE / THREADCOUNT + 1; *// 시작 값 [1, 26, 51, 76]*

(정적 할당으로 롤백)

1. **console 가시성 증가**

테스트를 진행할 때, 프로세스는 번호가 잘 보이는데, 쓰레드는 무슨 프로세스의 쓰레드인지, 몇 번째 쓰레드인지 확인하기 어려웠습니다.

따라서 다음과 같이 코드를 추가하였습니다.

int args[5];

args[0] = process\_num \* 2 + 1; *// 곱할 값 [3, 5, 7, 9]*

args[2] = i \* RANGE / THREADCOUNT; *// 종료 값 [25, 50, 75, 100]*

args[1] = args[2] - RANGE / THREADCOUNT + 1; *// 시작 값 [1, 26, 51, 76]*

args[3] = process\_num; *// process num*

args[4] = i; *// thread num*

(args 인자 추가)

int mul\_value = \*((int\*)args);

int start = \*((int\*)args + 1);

int end = \*((int\*)args + 2);

int process\_num = \*((int\*)args + 3);

int thread\_num = \*((int\*)args + 4);

printf("[Process[%d]-thread[%d] start]\n", process\_num, thread\_num);

(쓰레드 시작과 끝 printf에 번호 출력)

폰트, 텍스트, 스크린샷, 그래픽이(가) 표시된 사진

자동 생성된 설명

1. **몇 프로세스가 실행되지 않는 문제**

문제 코드

void\* func(void\* *args*) {

    int mul\_value = \*((int\*)args);

    int start = \*((int\*)args + 1);

    int end = \*((int\*)args + 2);

    int process\_num = \*((int\*)args + 3);

    int thread\_num = \*((int\*)args + 4);

    printf("[Process[%d]-thread[%d] start]\n", process\_num, thread\_num);

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

        printf("%3d \* %3d = %3d\n", i, mul\_value, i \* mul\_value);

    }

    printf("[Process[%d]-thread[%d] end]\n", process\_num, thread\_num);

    pthread\_exit(NULL);

}

*// process task*

void process\_task(int *process\_num*) {

    pthread\_t tid[THREADCOUNT];

    printf("[Process %d start]\n", process\_num);

*// thread 생성*

*for* (int i = 1; i <= THREADCOUNT; i++) {

        int args[5];

        args[0] = process\_num \* 2 + 1; *// 곱할 값 [3, 5, 7, 9]*

        args[2] = i \* RANGE / THREADCOUNT; *// 종료 값 [25, 50, 75, 100]*

        args[1] = args[2] - RANGE / THREADCOUNT + 1; *// 시작 값 [1, 26, 51, 76]*

        args[3] = process\_num; *// process num*

        args[4] = i; *// thread num*

*if* (pthread\_create(&tid[i], NULL, func, &args) != 0) {

            fprintf(stderr, "thread create error\n");

            exit(1);

        }

    }

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

        pthread\_join(tid[i], NULL);

    }

    printf("[Process %d end]\n", process\_num);

}

위와 같이 코딩할 경우, 아래 출력창과 같이 연산이 되지 않는 문제가 발생했습니다.

텍스트, 폰트, 스크린샷, 디자인이(가) 표시된 사진

자동 생성된 설명

(Process 2가 시작되었지만, 연산은 없는 상태로 종료됨.)

이 문제를 해결하기 위한 첫 방법은 func 함수를 여러 개 만드는 것이었습니다.

(func1, func2, func3, func4 정의함)

void\* (\*thread\_funcs[THREADCOUNT])(void\*) = { func1, func2, func3, func4 };

*if* (pthread\_create(&tid[i], NULL, thread\_funcs[i], &args) != 0) {

하지만 효과가 없었고, 제 첫 방법이 틀렸다는 것을 구글링을 통해 알게 되었습니다.

코드는 다시 되돌린 후, 출력 창을 보며 진정한 문제를 찾았습니다.

문제는 1~25까지의 연산이 건너뛰어지고 있었던 것이었습니다.

따라서, 문제가 있던 process\_task 함수를 수정하였습니다.

문제 코드

void process\_task(int *process\_num*) {

    pthread\_t tid[THREADCOUNT];

    printf("[Process %d start]\n", process\_num);

*// thread 생성*

*for* (int i = 1; i <= THREADCOUNT; i++) {

        int args[5];

        args[0] = process\_num \* 2 + 1; *// 곱할 값 [3, 5, 7, 9]*

        args[2] = i \* RANGE / THREADCOUNT; *// 종료 값 [25, 50, 75, 100]*

        args[1] = args[2] - RANGE / THREADCOUNT + 1; *// 시작 값 [1, 26, 51, 76]*

        args[3] = process\_num; *// process num*

        args[4] = i; *// thread num*

*if* (pthread\_create(&tid[i], NULL, thread\_funcs[i], &args) != 0) {

            fprintf(stderr, "thread create error\n");

            exit(1);

        }

    }

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

        pthread\_join(tid[i], NULL);

    }

    printf("[Process %d end]\n", process\_num);

}

수정된 코드

*// process task*

void process\_task(int *process\_num*) {

    pthread\_t tid[THREADCOUNT];

    printf("[Process %d start]\n", process\_num);

*/// thread 생성*

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

        int args[5];

        args[0] = process\_num \* 2 + 1; *// 곱할 값 [3, 5, 7, 9]*

        args[2] = (i + 1) \* RANGE / THREADCOUNT; *// 종료 값 [25, 50, 75, 100]*

        args[1] = args[2] - RANGE / THREADCOUNT + 1; *// 시작 값 [1, 26, 51, 76]*

        args[3] = process\_num; *// process num*

        args[4] = i + 1; *// thread num*

*if* (pthread\_create(&tid[i], NULL, func, args) != 0) {

            fprintf(stderr, "thread create error\n");

            exit(1);

        }

    }

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

        pthread\_join(tid[i], NULL);

    }

    printf("[Process %d end]\n", process\_num);

}

해당 방법으로

텍스트, 스크린샷, 메뉴이(가) 표시된 사진

자동 생성된 설명

위와 같은 출력 결과를 얻을 수 있었습니다.

1. **start, end 값 문제**

start 변수와 end 변수가 제대로 된 값을 가지지 못하는 문제가 있었습니다.

예를 들어, Process1의 thread1이 1~25의 값을 연산해야 하지만,

thread4의 값인, 76~100의 값을 연산하던 문제였습니다.

이 문제는 start, end 값을 정적으로 할당해서 생긴 문제였습니다.

이전 문제 해결 방법으로 동적 할당을 했다가 롤백하였는데, 다시 동적 할당으로 변경하였습니다.

*// 각 스레드마다 args 배열을 동적으로 할당 + 초기화*

int\* args = (int \*)malloc(5 \* sizeof(int));

이후

텍스트, 스크린샷, 폰트, 디자인이(가) 표시된 사진

자동 생성된 설명

정상적인 값을 확인할 수 있었습니다.

1. **ㅇㄹ**