PROJ5 보고서

소프트웨어학부 2019007329 이지현

1. 알고리즘 설명

- pthread_pool.h

pthread_pool_t struct에 running 변수를 bool이 아닌 int형으로 바꿔주었다. 기본 실행상태를 0, POOL_COMPLETE 종료 상태를 1, POOL_DISCARD 종료 상태를 2로 한다.

- worker

스레드가 작업을 수행할 함수이다. 스레드 풀을 생성하여 worker의 param 파라미터를 할당한다. 즉시 종료하는 옵션 POOL_DISCARD인 pool->running == 2가 발생하는 상황을 제외한 pool->running < 2일 경우 while문을 계속해서 반복한다. while문을 들어오면 대기열 접근을 위해 pool->mutex 락을 건다. 종료 명령이 없고(pool->running == 0) 대기열에 작업이 없는 경우(pool->q_len == 0) while문을 돌며 pool->full조건변수가 pthread_cond_wait으로 대기하도록 한다. while문을 빠져나온 뒤, 그 후 종료 명령이 발생했을 수도 있기 때문에 조건문으로 종료 명령 상태를 확인한다. running == 2이면, 락을 푼 뒤 pthread_exit(NULL)로 바로 종료한다. running == 1이고 대기열의 길이가 0이면 COMPLETE 종료 옵션이 발생했고 대기열에 있는 모든 스레드가 실행이 완료되었다는 의미이므로 마찬가지로 락을 푼 뒤 종료한다. 종료 명령이 없다면 작업을 진행한다. task_t 타입의 task에 (pool->q)[pool->q_front]로 버퍼의 q_front 인덱스에 있는 작업을 꺼낸다. 작업이 하나 빠져나왔으니 q_len을 하나 줄여주고, task.function(task.param)으로 작업을 실행한다. 작업을 실행시켰으니 empty 조건변수에 signal을 보내고 락을 해제한다.

- pthread_pool_init

스레드풀을 생성하는 함수이다. 일꾼 스레드의 크기와 대기열의 크기가 max를 넘으면 POOL_FAIL을 return하고 일꾼 스레드의 수보다 버퍼의 용량이 작으면 버퍼의 용량을 일꾼 스레드로 조정한다. 일꾼 스레드와 대기열을 malloc을 이용해 공간을 할당해준다. 그 후 pthread_pool_t 구조체의 모든 변수를 초기화시킨다. running은 기본 실행값인 0으로, q_size는 사용자가 요청한 버퍼 용량인 queue_size로, q_front와 q_len은 현재 버퍼가 비어있고 실행이 되지 않은 상태이므로 0으로, 일꾼 스레드도 사용자가 요청한 bee_size로 값을 할당한다.

일꾼 스레드를 동기화시킬 용도로 사용할 mutex, full, empty를 pthread_mutex_init, pthread_cond_init으로 초기화 한 후, 일꾼 스레드를 pthread_create로 생성한다. 생성한 pool->bee[i] 스레드가 worker 함수의 파라미터로 pool을 넘겨주며 실행하도록 설정하였다. 모든 과정을 마치면 POOL_SUCCESS를 return한다.

- pthread_pool_submit

스레드풀에서 실행시킬 함수와 인자의 주소를 넘겨주며 작업을 요청하는 함수이다. 대기열 접근을 위해 락을 건 뒤, 대기열이 꽉 찼을때(q_len == q_size) flag가 POOL_NOWAIT이라면 기다리지 않고 바로 락을 푼 뒤 POOL_FULL을 return한다. flag == POOL_WAIT인 경우 대기열이 빌때까지 while문을 반복하며 empty 조건 변수를 wait 시킨다. 작업 함수를 설정하기 위해 task_t 타입의 task를 정의하고, task.function으로 사용자가 제공한 f를 할당하고, task.param으로 p를 할당한다. 대기열에 작업을 넣기 위해 작업을 넣을 위치 int tsak_back을 (pool->q_front + pool->q_len) % pool->q_size로 설정한다. 다음에 실행될 위치와 대기열의 길이를 더하면 작업이 들어있는 대기열의 가장 끝 인덱스를 구할 수 있다. q[task_back]에 task를 할당하고, 작업을 하나 넣었기 때문에 q_len을 하나 늘린다. 작업 요청이 끝났으므로 조건변수 full에 signal을 보내고 락을 해제한 뒤 POOL_SUCCESS를 return한다.

- pthread_pool_shutdown

스레드풀을 종료하는 함수이다. 대기열 접근을 위해 락을 걸고 how == POOL_COMPLETE이면 running을 1로 바꾼다. 그 후 모든 대기중인 full, empty 조건변수를 깨우기 위해 pthread_cond_broadcast 사용한다. how == POOL_DISCARD이면 running을 2로 바꾸고 마찬가지로 broadcast로 대기중인 모든 full, empty를 깨운다. 대기열 접근이 끝났으니 락을 해제하고 일꾼스레드 bee를 pthread_join 한다. 마지막으로 pthread_mutex_destroy로 mutex, full, empty를 반납하고, bee와 q에 할당한 공간 또한 free로 해제한다. 마지막으로 POOL_SUCCESS를 return한다.

2. 컴파일

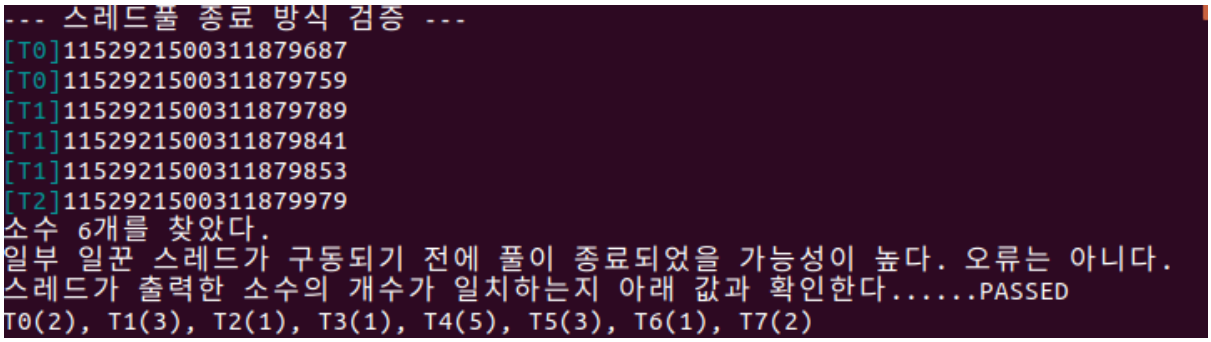
```
os@os-VirtualBox:~/Documents/theory_assignment/proj5$ make
gcc -o client client.o pthread_pool.o  -lpthread
```

3. 실행 결과

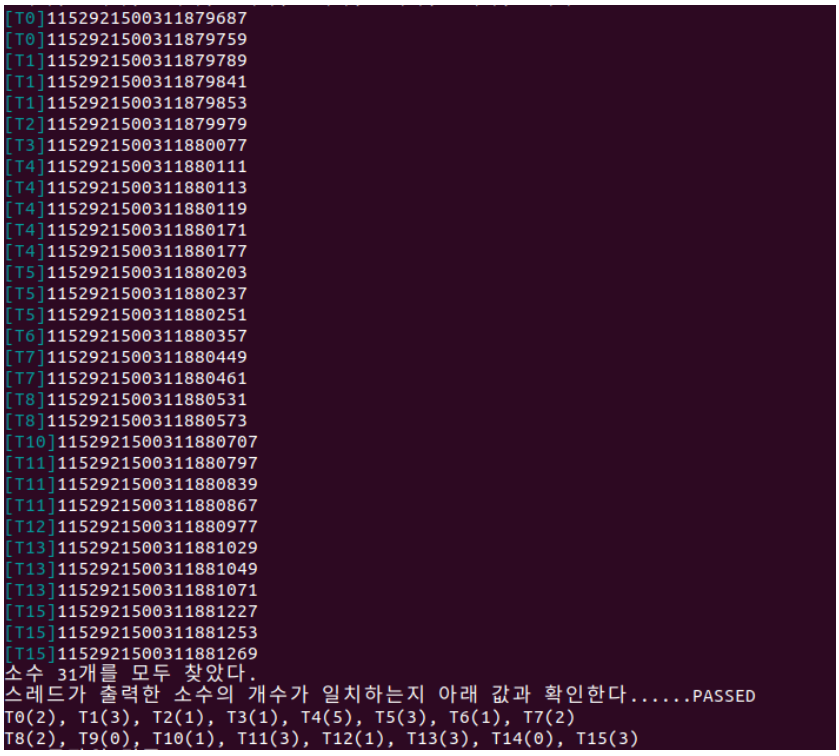
```
os@os-VirtualBox:~/Documents/theory_assignment/proj5$ ./client
--- 스레드풀 파라미터 한계 검증 ---
pthreaad_pool_init(): 일꾼 스레드 최대 수 초과.....PASSED
pthreaad_pool_init(): 대기열 최대 용량 초과.....PASSED
--- 스레드풀 최기화와 종료 검증 ---
pthread_pool_init(): 완료......PASSED
pthread_pool_shutdown(): 완료......PASSED
pthread_pool_init(): 완료......PASSED
pthread_pool_shutdown(): 완료......PASSED
--- 스레드풀 기본 동작 검증 ---
<0><1><12><13><14><15><16><17><18><19><20><21><22><23><24><25><26><27><28><29><
30><31><32><33><34><35><36><37><38><39><40><41><42><43><44><45><46><47><48><49>
<50><51><52><53><54><55><56><57><58><59><60><61><62><63><2><3><4><5><6><7><8><9
><10><11>[0][1][2][3][4][5][6][7][8][9][10][11][12][13][14][15][16][17][18][19]
[20][21][22][23][24][25][26][27][28][29][30][31][32][33][34][35][36][37][38][39
][40][41][42][43][44][45][46][47][48][49][50][51][52][53][54][55][56][57][58][5
9]......PASSED
```

스레드풀을 초기화하는 pthread_pool_init 함수에 일꾼 스레드와 대기열이 최대 수, 용량을 초과했

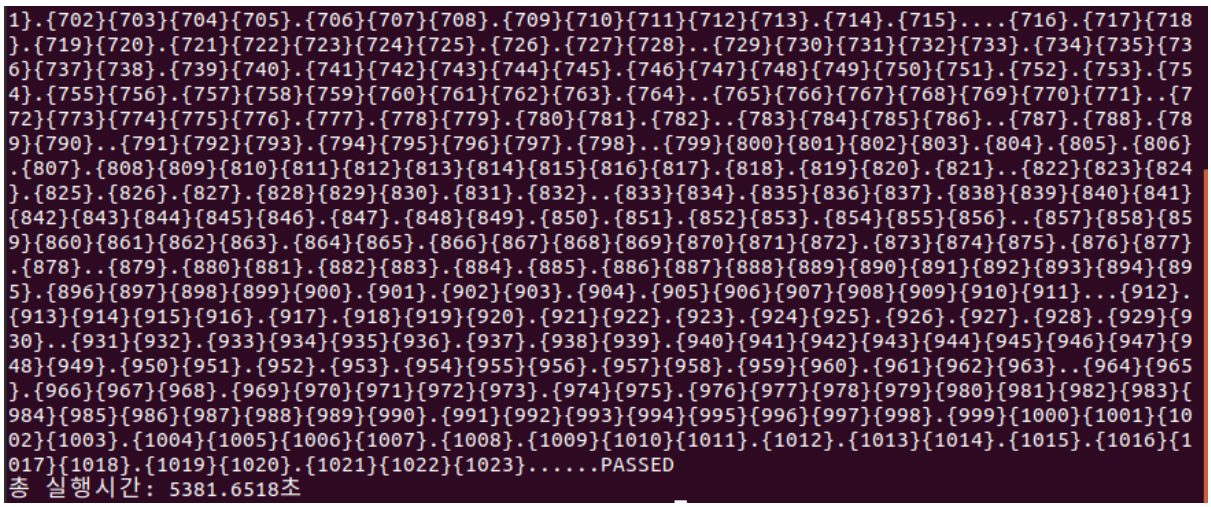
을 때 POOL_FAIL을 제대로 return한 결과를 보인다. 또한 초기화 함수 init과 종료 함수 shutdown이 POOL_COMPLETE, POOL_DISCARD 종료 옵션에 맞게 제대로 종료된 모습을 보인다. 그 후 대기열이 차서 기다리지 않고 거절된 작업을 빨간색으로 출력하는 기본 동작을 제대로 검증한 모습이다.



스레드 풀 종료 요청이 POOL_DISCARD 옵션으로 들어왔기 때문에 대기열에서 기다리던 스레드가 종료되고 소수를 6개 찾은 뒤 끝난 모습이다.



스레드풀 종료 요청이 POOL_COMPLETE 옵션으로 들어왔기 때문에 소수 31개를 모두 출력한 뒤 종료된 모습이다.



스레드풀이 잘 작동하는지 무작위로 생성해서 검증한다. 0부터 1023까지의 숫자가 랜덤한 dot과 함께 출력되는 것을 볼 수 있다. 총 실행시간은 5381.6518초이다.

4. 문제점과 느낀점

어려웠던 부분은 스레드풀 종료 방식이 두가지였던 점이었다. 종료 명령을 받았을 때, worker함수에서 바로 종료하느냐, 대기열에 있는 모든 작업을 실행한 뒤 종료하느냐에 따라 while문을 벗어나고 while문을 계속해서 반복하는지가 나뉘기 때문에 end_flag를 전역변수로 정의한 뒤 구분점 역할로 사용하였다. 그러나 무작위 검증을 하였을 때, 숫자가 1023까지 실행되지 못하는 결과를

얻게 되었다. 이 부분에 대해 교수님께 여쭤본 결과 전역변수로 flag를 설정하게 되면 스레드마다 다른 flag값을 가질 수도 있다는 조언을 얻을 수 있었다. 나는 flag를 POOL_COMPLETE로 설정하였지만 어떤 스레드는 다른 상태의 flag로 실행될 수 있다는 것이다. 교수님의 조언을 바탕으로 pthread_pool.h를 수정하여 pthread_pool_t 구조체에 bool running으로 사용하던 변수를 int running으로 바꿔주고, 기본 실행, POOL_COMPLETE, POOL_DISCARD 세 상태로 구분하였다. 이렇게 정의한 뒤 how에 따라 pool->running 값을 할당해 주었더니, 옳은 출력 결과를 얻을 수 있었다. 스레드를 관리할 때 가장 중요한 부분인 동기화를 놓친다면 큰 문제가 발생한다는 점을 알 수 있었고, 앞으로도 멀티스레드를 다루게 된다면 동기화 작업을 절대 잊지 말아야겠다고 다짐하게 된 계기가 되었다.