Operating Systems Project #5

과목명.	운영체제론
담 당.	오 희 국 교수님
제출일.	2023년 06월 05일
소프트웨어융합대학	컴퓨터학부
3 학년,	학 번. 2019033936
이 름.	이승섭

HANYANG UNIVERSITY

스레드풀 알고리즘

pthread_pool_init() 함수 설명 : 스레드풀을 초기화 시켜주는 함수로 일꾼 스레드와 대기열에 필요한 공간을 할당해주고 각각의 변수들을 초기화 시킨다. running 의 경우에는 true로 초기화 시켜주고 q 는 대기열에 필요한 공간을 할당해준다. q_size는 함수를 실행할 때 파라미터로 받아오는 queue_size로 초기화 시켜주고 q_front 와 q_len 은 맨 처음의 경우 대기열에 아무것도 없으므로 0으로 초기화 시켜준다. bee 의 경우 일꾼 스레드에 해당하므로 필요한 공간을 할당해준다. bee_size도 함수의 파라미터로 받아오는 bee_size로 초기화 시켜주고 상호베타 락과 조건변수도 init() 함수를 통해 초기화 시켜준다. 스레드풀 구조체 초기화가 끝났으므로 스레드풀 생성에 실패한 경우 POOL_FAIL을 리턴해준다. 조건으로 대기열에 필요한 공간을 할당하지 못했거나 일꾼 스레드에 필요한 공간을 할당하지 못했을 때 POOL_FAIL을 리턴해주고 또한 대기열과 일꾼 스레드에 할당된 공간이 지정된 공간보다 크면 그 때도 스레드풀 생성에 실패했으므로 POOL_FAIL을 리턴해준다. 그 다음 대기열에 사용할 원형 버퍼의 용량이 일꾼 스레드보다 작으면 효율을 극대화할 수 없기 때문에 원형 버퍼의 용량을 일꾼 스레드의 용량으로 상향 조정 시켜준다. 이 과정까지 끝내주면 스레드풀 생성에 성공한 것이므로 일꾼 스레드를 생성해주고 POOL_SUCCESS를 반환해준다. 여기까지가 pthread_pool_init() 함수 알고리즘 설명이다.

worker() 함수 설명 : 스레드풀에 있는 일꾼 스레드가 수행하는 함수로 작업이 있으면 FIFO 대기열에서 작업을 하나씩 꺼내서 실행하고 대기열에 작업이 없으면 새 작업이 들어올 때까지 기다린다. 우선 헤더파일에 있는 구조체의 포인터를 가져온 뒤 무한루프에 들어가는데 이 때 상호베타를 위해 뮤텍스락을 걸어준다. 스레드풀이 실행중이며 대기열이 비어 있으면 작업이 들어올 때까지 대기하고 스레드풀이 종료되면 뮤텍스락을 풀어주고 스레드를 종료한다. 작업이 들어오면 q_front를 1 증가시켜 다음 작업으로 이동하고 대기열에 있는 작업의 개수를 1만큼 감소한다. 만약 q_len의 값이 q_size의 값보다 작으면 대기열이 꽉 차 기다리고 있던 스레드를 깨워준다. 이 과정이 끝나면 뮤텍스락을 풀어주고 작업을 실행한다.

pthread_pool_submit() 함수 설명 : 스레드풀에서 실행 시킬 함수와 인자의 주소를 넘겨주며 작업을 대기열에 넣어주는데 flag의 값에 따라 방식이 달라지는 함수이다. 상호베타를 위해 먼저 뮤텍스락을 걸어주고 스레드풀의 대기열이 가득 찬 상황에서 flag의 값이

POOL_NOWAIT이면 기다리지 않고 나오기 때문에 POOL_FULL 을 리턴해주고 함수를 종료시켜주는데 이때 종료하기 전에 뮤텍스락을 풀어 주어 데드락에 걸리는 상황을 피해준다. flag의 값이 POOL_WAIT 이면 대기열에 빈 자리가 나올 때까지 대기시켜주며 일꾼 스레드에서 대기열에 빈 자리가 나왔다고 신호를 보내주면 그 때 반복문에서 나오게 된다. 반복문에서 나오면 대기열에 작업을 추가해주고 q_len의 값을 1 증가시켜준다. 또한 요청한 작업을 대기열에 넣어줬으므로 일꾼 스레드에서 대기열이 비어있을 때 대기 중인 스레드에 신호를 보내 깨워주고 뮤텍스락을 풀어준다. 작업 요청에 성공했으므로 POLL_SUCCESS를 반환해주고 함수를 종료한다.

pthread_pool_shutdown() 함수 설명 : 스레드풀을 종료하는 함수로 일꾼 스레드가 현재 작업 중이면 작업이 마치는 대로 종료 시켜주며 how 의 값에 따라 종료 시켜주는 방식이 달라진다. 부모 스레드는 종료된 일꾼 스레드와 조인한 뒤 자원을 반납한다. 먼저 상호베타를 위해 뮤텍스 락을 풀어주고 running 을 false로 바꿔주어 스레드풀을 종료한다. 그 다음 일꾼 스레드가 현재 작업중이면 그 작업을 마치게 한다. how의 값이 POOL_COMPLETE이면 대기열에 남아 있는 모든 작업을 마치고 종료하기 때문에 q_len 이 0보다 크면 대기열에 작업이 있다는 의미로 작업을 모두 실행해주고 반복문을 빠져나오며 how의 값이 POOL_DISCARD 이면 대기열을 비워주면서 종료해준다. 종료되었으면 종료된 일꾼 스레드와 조인 한 후에 뮤텍스와 조건 변수를 해제하고 할당된 공간도 해제하면서 자원을 반납한다. 종료가 완료되었으므로 POOL_SUCCESS를 반환해준다.

컴파일 화면을 보여주는 화면 캡처

실행 결과물 상세 설명

```
iseungseob@iseungseob-ui-MacBookAir 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 ---
<10><12><13><14><15><16><17><0><2><3><4><5><6><7><8><9><11><1><19><20><21><22><18><24><35><36><37><38><39><41><42><43><44><45><26
><33><28><29><30><31><32><33><34><40><46><47><27><25><48><60><61><62><63><49><50><52><53><54><55><56><57><58><59><51>[0][1][3][12
][4][6][7][8][9][10][11][5][13][12][15][14][16][17][19][20][21][18][32][23][24][26][35][27][29][30][38][31][33][34][35][36][37][3
8][39][40][41][42][43][44][45][46][52][47][49][50][51][52][53][54][55][56][48][57][58]......PASSED
--- 스레드풀 종료 방식 검증 ---
[T0]1152921500311879687
[T1]1152921500311879789
[T0]1152921500311879759
[T1]1152921500311879841
[T1]1152921500311879853
소수 5개를 찾았다.
일부 일꾼 스레드가 구동되기 전에 풀이 종료되었을 가능성이 높다. 오류는 아니다.
스레드가 출력한 소수의 개수가 일치하는지 아래 값과 확인한다......PASSED
T0(2), T1(3), T2(1), T3(1), T4(5), T5(3), T6(1), T7(2)
[T0]1152921500311879687
[T1]1152921500311879789
[T2]1152921500311879979
[T0]1152921500311879759
[T1]1152921500311879841
[T3]1152921500311880077
[T1]1152921500311879853
[T4]1152921500311880111
[T4]1152921500311880113
[T4]1152921500311880119
[T4]1152921500311880171
[T4]1152921500311880177
[T5]1152921500311880203
[T5]1152921500311880237
[T5]1152921500311880251
[T6]1152921500311880357
[T7]1152921500311880449
[T7]1152921500311880461
[T8]1152921500311880531
```

```
[T4]1152921500311880171
[T4]1152921500311880177
[T5]1152921500311880203
[T5]1152921500311880237
[T5]1152921500311880251
[T6]1152921500311880357
[T7]1152921500311880449
[T7]1152921500311880461
[T8]1152921500311880531
[T8]1152921500311880573
[T10]1152921500311880707
[T11]1152921500311880797
[T11]1152921500311880839
[T11]1152921500311880867
[T12]1152921500311880977
[T13]1152921500311881029
[T13]1152921500311881049
[T13]1152921500311881071
[T15]1152921500311881227
[T15]1152921500311881253
[T15]1152921500311881269
소수 31개를 모두 찾았다.
스레드가 출력한 소수의 개수가 일치하는지 아래 값과 확인한다......PASSED
T0(2), T1(3), T2(1), T3(1), T4(5), T5(3), T6(1), T7(2)
T8(2), T9(0), T10(1), T11(3), T12(1), T13(3), T14(0), T15(3)
--- 무작위 검증 ---
{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}{3
2}{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}{64}{65}{66}{67}{68}{69}{70}{71}{72}{73}.{74}{75}{76}{77}..{78}{79}{80}.{81}.{82}.{83}..{84}{85}.{86}.{87}{88}{89}{90}{91}{92}
{93}..{94}.{95}{96}{97}.{98}.{99}{100}.{101}..{102}{103}{104}{105}{106}{107}{108}.{109}.{110}..{111}.{112}.{113}.{114}.{115}{116}
{117}{118}{119}.{120}{121}..{122}.{123}{124}{125}..{126}{127}{128}{129}.{130}{131}{132}{133}...{134}{135}{136}{137}{138}.{139}.{1
40}{141}.{142}..{143}{144}{145}{146}{147}{148}{149}{150}{151}{152}{153}{154}{155}.{156}{157}{158}.{159}{160}{161}{162}...{163}{16
4}{165}{166}{167}.{168}{169}{170}{171}{172}{173}{174}.{175}{176}.{177}{178}.{179}{180}.{181}{182}{183}..{184}{185}{186}{187}.{188
}{189}{190}{191}{192}.{193}.{194}.{195}{196}{197}{198}{199}{200}{201}.{202}{203}.{204}...{205}.{206}{207}{208}{209}{210}{211}.{21
2}.{213}...{214}{215}{216}{217}..{218}{219}.{220}.{221}{222}.{223}{224}{225}.{226}.{227}{228}.{229}.{230}{231}.{232}.{233}{234}..
{235}{236}{237}{238}{239}.{240}{241}{242}{243}{244}.{245}..{246}.{247}{248}.{249}{250}.{251}.{252}.{253}{254}{255}{256}{257}.{258
}{259}.{260}{261}{262}{263}{264}.{265}{266}{267}{268}{269}{270}{271}{272}{273}{274}{275}{276}.{277}{278}..{279}..{280}{281}{282}{
283}..{284}{285}.{286}{287}{288}{289}..{290}{291}.{292}{293}{294}{295}{296}{297}.{298}{299}{300}{301}{302}{303}.{304}{305}{306}{3
07}.{308}{309}..{310}{311}{312}{313}.{314}{315}..{316}.{317}{318}{319}{320}{321}{322}{323}.{324}..{325}{326}.{327}{328}{329}{330}
{331}.{332}{333}{334}{335}{336}{337}{338}.{339}..{340}.{341}{342}.{343}{344}..{345}{346}{347}{348}.{349}{350}{351}.{352}{353}{354
}{355}{356}{357}{358}{359}{360}{361}{362}..{363}{364}.{365}{366}{367}{368}.{369}{370}..{371}.{372}..{373}.{374}{375}{376}{377}{37
8}{379}{380}.{381}.{382}{383}{384}.{385}..{386}.{387}{388}{389}{390}.{391}{392}.{393}{394}.{395}{396}{397}{398}{399}{400}.{401}.{
402}{403}{404}.{405}{406}.{407}{408}{409}{410}{411}{412}{413}{414}{415}{416}{417}{418}.{419}{420}.{421}..{422}.{423}.{424}{425}.{
```

스레드풀이 잘 구현되었는지 확인하기 위해 client.c의 메인 함수에서 이를 검증하는 결과이다. 첫번째 검증으로 스레드풀의 한계를 검사하는데 일꾼 스레드의 최대 개수와 대기열 최대 용량을 초과했다고 출력하는데 pthread_pool_init() 함수에서 정해진 용량보다 초과했을때 POOL_FAIL을 리턴해주기 때문에 이와 같이 출력이 잘 됐음을 알 수 있다. 그 다음으로 스레드풀이 정상적으로 초기화되고 출력되는지 검사하는데 스레드풀을 초기화하고 종료할 때 POOL_DISCARD 인 경우와 POOL_COMPLETE 인 경우 2가지를 검사하며 오류 없이 잘 출력됨을 결과를 통해 볼 수 있다.

그 다음으로 스레드풀의 기본 동작을 검증하는데 submit() 함수에서 POOL_NOWAIT 인 경우 대기열의 길이가 짧아서 처리할 수 없는 요청이 발생할 수도 있는데 대기열이 차서 거절된 작업을 빨간색으로 표시하여 63까지 빨간 색과 흰색이 번갈아 가면서 출력이 잘 되고 있음을 알 수 있고 POOL_WAIT 인 경우 대기열에 빈 자리가 날 때까지 기다리기 때문에 순서대로 잘 출력 되고 있음을 알 수 있다.

그 다음으로는 스레드풀의 종료 방식을 검증하는데 비효율적으로 소수을 찾는 작업을 스레드풀에 요청하고 종료를 POOL_DISCARD 방식으로 종료하기 때문에 소수 5개만 찾고 종료했음을 알 수 있다. 또한 POOL_COMPLETE 방식으로 종료하는데 이 때는 소수 31개를 모두 찾고 종료함으로서 검증이 잘 되는 것을 확인할 수 있다.

마지막으로 무작위 검증을 하는데 스레드풀 3개를 무작위로 생성한 뒤 점과 숫자를 출력하는 방식으로 검증하며 마지막으로 스레드풀이 종료됐을 때 1023 까지 출력되고 총 실행시간이 나오며 종료되는 모습을 확인할 수 있다.

과제를 수행하면서 경험한 문제점과 느낀점

우선 스레드풀에 대한 개념과 어떤 방식으로 생성하고 사용하고 없애는지 자료만 보고 이해하기 힘들어서 검색을 통해 개념과 작동 방식을 배웠고, 초반에 뼈대 코드가 아예 없어 c언어 구조체 및 포인터를 사용하는데 시간이 좀 오래 걸렸다. 또한 pthread_pool_init() 과 pthread_pool_submit() 까지는 생각보다 구현이 빠르게 됐는데 worker() 함수와 특히 pthread_pool_shutdown() 을 구현하는데 개념 정리가 확실히 되지 않아 시간을 오래 허비했다. 그리고 코드를 완성하고 실행 시켰을 때 스레드풀 기본 동작 검증에서 데드락이 걸리는 상황을 겪었는데 pthread_pool_submit() 함수에 flag 가 NO_WAIT 인 경우 POOL_FULL 을 리턴해주며 함수를 종료하는데 이 때 뮤텍스 락을 풀어 주지 않아 발생한 상황이였다. 또한 기본 동작 검증이 끝나고 스레드풀 종료 방식 검증에서 pthread_pool_shutdown() 함수의 how 가 POOL_COMPLETE 인 경우 소수 31개를 출력해줘야 하는데 31개를 출력해주지 못하는 상황도 겪었는데 반복문과 조건문, pthread_cond_broadcast(&pool->full) 코드를 통해 대기열에 남아 있는 모든 작업을 마치고 종료해줄 수 있었다. 이번 기회를 통해 스레드풀을 처음 접했는데 굉장히 신선한 경험이라고 생각했으며 스레드의 생성과 소멸에 의한 비효율적인 측면을 없애고자 만들어진 프로그래밍 기법으로 사용한 다는 사실을 깨달을 수 있었다.