

Operating System Project #4

2017011885 김현기

2017011903 남윤재

2019040519 김태형

1. 스레드풀 알고리즘 구현

(1) pthread_pool_init

먼저 스레드 풀을 초기화 하기 위해 들어온 인자를 확인한다. bee_size가 POOL_MAXBSIZE 보다 크거나, bee_size가 0보다 작거나 같거나, queue_size가 POOL_MAXQSIZE보다 크거나, queue_size가 0보다 작거나 같으면 즉시 POOL_FAIL를 리턴한다. 문제 조건에 맞게 bee_size가 queue_size보다 작은 값이 들어온 경우에는 queue_size를 bee_size와 동일하게 초기화 해준다.

pool 구조체의 변수 running, q_front, q_len, bee_size 또한 각각 true, 0, bee_size로 초기화 해주고, 대기열 q와 일꾼 스레드 bee를 동적할당 해준다. 마찬가지로 뮤텍스락과 조건변수 2개도 각각 초기화 해준다.

이후, 일꾼 스레드 worker를 bee_size만큼 생성해준다. 각 일꾼 스레드들에게는 구조체 pool을 인자로 전달한다. 스레드 생성 과정 중에 오류가 발생하면, pthread_pool_shutdown 함수를 실행해 만들어놓은 스레드 풀을 shutdown 시키고, POOL_FAIL을 리턴한다. 오류가 없는 경우, POOL_SUCCESS를 리턴한다.

(2) worker

worker함수는 pool 내부에 있는 lock과 task_t에 접근해야 하기 때문에 인자를 pool로 받아야한다. running 변수가 true일 동안 While 루프를 돌며 대기열에서 작업을 하나씩 가져와 실행한다.

먼저 mutex를 획득하고 대기열이 비어있을 경우, while문을 이용해 조건 변수 full안에서 새 작업이 들어올 때까지 기다린다. 대기가 종료되었다면 pool의 running이 FALSE로 바뀌었을 경우를 탐지해 루프를 종료해야 교착상태나 포인터 오류를 방지할 수 있으므로 running 조건을 재검사한다.

이후, 대기열에서 task_t 하나를 빼서 fnc에 실행할 함수와 인자를 저장하고, 대기열 실행 위치 q_front를 1칸 밀어주고, q_len을 1 감소시킨다. 이후, mutex를 해제한 뒤 대기열에 빈자리가 있음을 signal을 보내 알리고, 저장한 함수를 실행한다.

(3) pthread_pool_submit

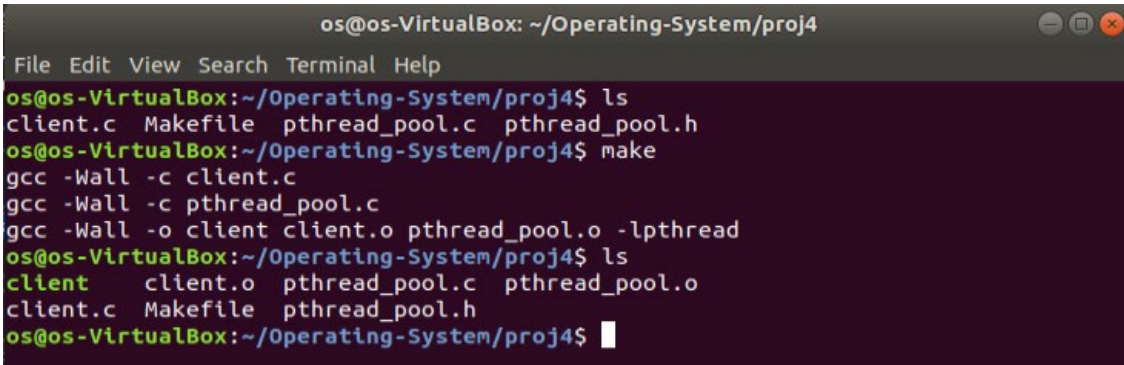
먼저 뮤텍스를 획득하고, 대기열의 상태를 확인한다. 대기열이 가득 차있지 않다면 새 작업을 원형 대기큐에 넣고, 대기열의 길이를 하나 증가시킨다. 대기열이 가득 차 있고 flag값이 POOL_NOWAIT이라면, 뮤텍스를 해제하고 즉시 POOL_FULL을 리턴한다. 대기열이 가득 차있고, flag값이 POOL_WAIT라면 while문을 사용해 대기열에 빈자리가 생길때까지 조건변수 empty에서 대기한다. 대기가 종료되고, running의 값이 바뀌어 false가 되었다면 스레드를 종료시킨다. 문제가 없다면 새 작업을 대기큐에 넣어준다. 이후 뮤텍스를 풀고, 조건변수 full에 signal을 보내주고 함수를 종료한다.

(3) pthread_pool_shutdown

뮤텍스를 걸고, pool 구조체의 running 값을 false로 바꾸어 실행중인 일꾼 스레드들이 자연스럽게 루프를 빠져나오도록 한다. pthread_cond_broadcast로 조건변수 empty와 full에서 기다리고 있는 일꾼 스레드들을 깨우고, pthread_join 함수로 모든 일꾼 스레드들을 종료시킨다.

이후, 동적할당 시켜주었던 일꾼 스레드 bee와 대기열 q를 free 함수로 메모리 해제시키고, 뮤텍스와 조건변수 2개도 각각 destroy 해준다. 이후, POOL_SUCCESS를 리턴한다.

2. 컴파일 과정

A terminal window titled 'os@os-VirtualBox: ~/Operating-System/proj4' showing the compilation of a C program. The user runs 'ls' showing 'client.c', 'Makefile', 'pthread_pool.c', and 'pthread_pool.h'. Then they run 'make', which executes 'gcc -Wall -c client.c', 'gcc -Wall -c pthread_pool.c', and 'gcc -Wall -o client client.o pthread_pool.o -lpthread'. A second 'ls' command shows the resulting files: 'client', 'client.o', 'pthread_pool.c', and 'pthread_pool.o'.

```
os@os-VirtualBox: ~/Operating-System/proj4
File Edit View Search Terminal Help
os@os-VirtualBox:~/Operating-System/proj4$ ls
client.c Makefile pthread_pool.c pthread_pool.h
os@os-VirtualBox:~/Operating-System/proj4$ make
gcc -Wall -c client.c
gcc -Wall -c pthread_pool.c
gcc -Wall -o client client.o pthread_pool.o -lpthread
os@os-VirtualBox:~/Operating-System/proj4$ ls
client client.o pthread_pool.c pthread_pool.o
client.c Makefile pthread_pool.h
os@os-VirtualBox:~/Operating-System/proj4$
```

오류 메시지 없이, client, client.o, pthread_pool.o 파일이 생성된 것을 확인할 수 있다.

[illegible]

