

Operating Systems

Project #5

컴퓨터학부

2024년 5월 27일

스레드풀

스레드풀은 스레드의 생성과 관리에 필요한 사용자의 부담과 비용을 줄이고, 시스템의 스레드 수가 제한된 범위를 넘지 않도록 조절하는 유용한 방식이다. 사용자가 작업을 스레드풀에 제출하면 풀 안에 있는 스레드가 그 작업을 실행한다. 제출된 작업은 대기열에서 기다리다 한가한 스레드에 의해 선택되어 실행된다. 대기열에 더 수행할 작업이 없으면 스레드는 새 작업이 들어와 깨워줄 때까지 대기상태로 있다. 이 과제는 Pthread와 POSIX 조건변수를 사용하여 스레드풀을 구현하는 것이다.

클라이언트

사용자는 다음과 같은 API를 사용하여 스레드풀에 접근한다.

- `int pthread_pool_init(pthread_pool_t *pool, size_t bee_size, size_t queue_size)` – 스레드의 수가 `bee_size`이고, 대기열의 용량이 `queue_size`인 스레드풀 `pool`을 생성한다. `bee_size`는 시스템이 정한 `POOL_MAXBSIZE`를 넘을 수 없다. `queue_size`는 `POOL_MAXQSIZE`를 넘을 수 없다. 오류가 없으면 `POOL_SUCCESS`를, 그렇지 않으면 `POOL_FAIL`을 리턴한다.
- `int pthread_pool_submit(pthread_pool_t *pool, void (*f)(void *p), void *p, int flag)` – 스레드풀 `pool`에 작업을 제출한다. `f`는 작업으로 수행할 함수의 포인터(주소)이며, `p`는 `f` 함수에 넘겨주는 인자의 포인터이다. `flag`은 대기열에 빈 자리가 없을 때 기다릴 것인지 여부를 결정하는 값이다. `flag`의 값이 `POOL_WAIT`이면 대기열에 빈 자리가 생길 때까지 기다리다 제출하고, `POOL_SUCCESS`를 리턴한다. 반면에 대기열에 빈 자리가 없고 `flag`의 값이 `POOL_NOWAIT`이면, 기다리지 않고 즉시 `POOL_FULL`을 리턴한다. 스레드풀이 종료하기 위해 작업을 받지 않으면 `POOL_FAIL`을 리턴한다. 제출이 완료되면 `POOL_SUCCESS`를 리턴한다.
- `int pthread_pool_shutdown(pthread_pool_t *pool, int how)` – 새 작업을 받지 않고 스레드풀 `pool`을 종료한다. 일꾼 스레드가 현재 작업 중이면 그 작업을 마치게 한다. `how`의 값이 `POOL_COMPLETE`이면 대기열에 남아 있는 모든 작업을 마치고 종료한다. `POOL_DISCARD`이면 대기열에 작업이 남아 있어도 더 이상 수행하지 않고 종료한다. 종료가 완료되면 `POOL_SUCCESS`를 리턴한다.

이 과제에서는 위 함수를 활용한 샘플 프로그램인 `client.c`를 학생들에게 제공한다. `client.c`는 위에서 언급한 스레드풀의 기능이 잘 수행되는지 검증하는 프로그램으로 임의로 수정해서는 안 된다.

스레드풀 제어블록

스레드풀 제어블록은 생성에서 소멸까지 스레드풀을 운영하는데 필요한 여러 가지 정보를 담고 있다. 모든 스레드풀은 각자 고유의 제어블록을 가지고 있으며, 이 제어블록은 `pthread_pool_t` 타입으로 다음과 같이 정의한다.

```
typedef struct {
    int state;                /* 실행, 종료, 대기 등 스레드풀의 현재 상태 */
    task_t *q;               /* FIFO 작업 대기열로 사용할 원형 버퍼 */
    int q_size;              /* 원형 버퍼 q 배열의 크기 */
    int q_front;             /* 대기열에서 다음에 실행될 작업의 위치 */
    int q_len;               /* 대기열의 길이 */
    pthread_t *bee;          /* 일꾼(일벌) 스레드의 ID를 저장하기 위한 배열 */
    int bee_size;            /* bee 배열의 크기로 일꾼 스레드의 수를 의미 */
    pthread_mutex_t mutex;   /* 대기열을 접근하기 위해 사용하는 뮤텍스락 */
    pthread_cond_t full;     /* 빈 대기열에 새 작업이 들어올 때까지 기다리는 곳 */
    pthread_cond_t empty;    /* 대기열에 빈 자리가 발생할 때까지 기다리는 곳 */
} pthread_pool_t;
```

스레드풀에 상주하고 있는 스레드는 작업이 있으면 수행하고, 없으면 대기상태로 있으면서 스레드풀이 종료될 때까지 이 과정을 무한히 반복한다. 이 스레드를 우리는 편의상 일꾼(일벌) 스레드라고 부른다. 스레드풀의 현재 상태를 나타내는 `state`는 스레드풀이 실행 중인지, 아니면 종료하기 위해 대기열을 비우는 중인지, 아니면 종료되었는지 등 개발자의 필요에 따라 여러 가지 상황을 정의하고, 그 값에 따라 다음 할 일을 올바르게 결정하는데 활용한다.

스레드풀의 FIFO 작업 대기열로 사용할 배열 `q`는 원형 버퍼이다. `q_size`는 배열 `q`의 크기를 나타내며, 원형 버퍼의 용량을 의미한다. `q_front`는 대기열에서 다음에 실행될 작업의 위치로 배열 `q`의 색인 값이다. 대기열의 길이를 나타내는 `q_len`의 값이 0이면 현재 대기하고 있는 작업이 없다는 뜻이다. 반면에 `q_len`의 값이 `q_size`이면 대기열이 차서 더 이상 새 작업을 넣을 수 없는 상황을 의미한다.

`bee`는 작업을 수행하는 일꾼 스레드의 ID를 저장하는 배열이다. `bee_size`는 배열 `bee`의 크기를 나타내며, 이는 상주하는 일꾼 스레드의 개수를 의미한다. `full`과 `empty`는 대기열에 작업이 채워지기를 또는 빈 자리가 생기기까지 기다리는 POSIX 조건변수이다. `mutex`는 대기열을 조회하거나 변경하기 위해 사용하는 뮤텍스락으로 조건변수 `full`, `empty`와 연계해서 사용한다.

스레드풀 구현

스레드풀 구현에 필요한 일꾼 함수와 사용자 API 함수의 골격은 파일로 제공한다. 학생들은 다음에서 언급할 함수를 포함해서 필요하다고 판단되는 함수를 스스로 설계하고 구현한다. 다만 정보의 은닉화를 위해 세 개의 API 함수 이외의 내부 함수는 밖에서 보이지 않도록 키워드 `static`을 사용하여 숨긴다. 일꾼 스레드가 사용할 함수와 사용자 API 함수 구현에 대한 개괄적인 설명은 아래와 같다.

1. `static void *worker(void *param)` 함수는 풀에 있는 일꾼 스레드가 수행할 함수로 FIFO 대기열에서 작업을 하나씩 꺼내서 실행한다. 대기열에 작업이 없으면 새 작업이 들어올 때까지 기다린다. 이 과정을 스레드풀이 종료될 때까지 반복한다.
2. `pthread_pool_init()` 함수는 스레드풀을 초기화한다. 우선 사용자가 요청한 일꾼 스레드와 대기열에 필요한 공간을 할당하고 변수를 초기화한다. 일꾼 스레드의 동기화를

위해 사용할 뮤텍스락과 조건변수도 초기화한다. 마지막 단계에서는 일꾼 스레드를 생성하여 각 스레드가 `worker()` 함수를 실행하게 한다. 대기열로 사용할 원형 버퍼의 크기가 일꾼 스레드의 수보다 작으면 효율을 극대화할 수 없다. 사용자가 요청한 `queue_size`가 최소한 `bee_size` 되도록 자동으로 상향 조정한다.

3. `pthread_pool_submit()` 함수는 사용자가 요청한 작업을 대기열에 넣는다. 대기열이 꽉 찬 상황에서는 옵션에 따라 기다릴지 말지를 결정한다. 스레드풀이 종료 중이면 작업을 대기열에 넣지 말고 나온다.
4. `pthread_pool_shutdown()` 함수는 옵션에 따라 일꾼 스레드를 종료시킨다. 메인(부모) 스레드는 종료된 일꾼 스레드와 조인한 후에 스레드풀에 할당된 자원을 반납한다. 뮤텍스락과 조건변수는 초기화 후 다시 사용하므로 제거(`destroy`)하지 않는다. 스레드를 종료시키기 위해 철회 (`cancellation`)를 생각할 수 있으나, 이 경우에는 바람직하지 않다. 일꾼 스레드가 현재 진행 중인 작업을 마쳐야 하기 때문에 철회 지점이 문제가 된다. 뿐만 아니라 락을 소유한 스레드를 철회하면 교착상태가 발생하기 쉽다.

골격파일

`pthread_pool.skeleton.c`에는 구현에 필요한 최소한의 함수가 들어 있다. 학생들은 필요하다고 판단되는 함수를 추가할 수 있다. 스레드풀 제어블록과 상수가 정의되어 있는 헤더파일인 `pthread_pool.h`도 필요에 따라 수정할 수 있다. 그러나 표준 스레드풀 API만 사용하는 `client.c`를 수정해서는 안 된다. 골격파일과 함께 컴파일 과정을 쉽게 해주는 `Makefile`도 제공한다.

제출물

스레드풀이 잘 설계되고 구현되었다는 것을 보여주는 자료를 각자가 판단하여 PDF로 묶어서 이름_학번_PROJ5.pdf로 제출한다. 여기에는 다음과 같은 것이 반드시 포함되어야 한다.

- 본인이 설계한 스레드풀 알고리즘 (2쪽 분량)
- 컴파일 과정을 보여주는 화면 캡처
- 실행 결과물의 주요 장면을 발췌해서 그에 대한 상세한 설명
- 과제를 수행하면서 경험한 문제점과 느낀점
- 프로그램 소스파일 2개 (`pthread_pool.c`, `pthread_pool.h`) 별도 제출
- 프로그램 실행 결과물 (`client.txt`) 별도 제출

평가

- **Correctness 50%:** 프로그램이 올바르게 동작하는 지를 보는 것입니다. 여기에는 컴파일 과정은 물론, 과제가 요구하는 기능이 문제없이 잘 작동한다는 것을 보여줘야 합니다.
- **Presentation 50%:** 자신의 생각과 작성한 프로그램을 다른 사람이 쉽게 이해할 수 있도록 프로그램 내에 적절한 주석을 다는 행위와 같이 자신의 결과를 잘 표현하는 것입니다. 뿐만 아니라, 프로그램의 가독성, 효율성, 확장성, 일관성, 모듈화 등도 여기에 해당합니다. 이 부분은 상당히 주관적이지만 그러면서도 중요한 부분입니다. 컴퓨터과학에서 중요하게 생각하는 `best coding practices`를 참조하기 바랍니다.

HK