Project 03

Implementing light-weight process in xv6 and locking method

Due date

2024. 05. 19. 11:59 pm

Introduction

- XV6에서 Light Weight Process(LWP)를 구현하고 Linux에서 pthread 간 상호 배제를 보장하기 위한 locking method를 고안하는 것이 이번 프로젝트의 목표입니다.

Light-weight process

- 일반적으로 프로세스는 서로 독립적으로 실행되고, 자원을 공유하지 않으며 서로 별개의 주소 공간과 파일 디스크립터를 가집니다.
  xv6에는 본래 이러한 "프로세스"만이 구현되어 있습니다.

- Light-wight process (LWP)는 다른 LWP와 자원과 주소 공간 등을 공유하여
  유저 레벨에서는 멀티태스킹을 가능하게 해주는 개념입니다.

POSIX thread

- LWP를 구현하기 위해, 이번 과제에서는 간단한 POSIX thread (pthread)를 구현합니다.
- Pthread는 유닉스 계열의 운영체제에서 병렬적인 프로그램을 작성하는 데에 주로 사용되는 API로, LWP의 대표적인 구현체 중 하나입니다.
- Pthread의 기본적인 기능을 직접 구현해 보면서 LWP의 동작 원리를 이해하는 것이 이번 과제의 목표입니다.

Specification - API (thread_create)

- *int thread_create(thread_t *thread, void *(*start_routine)(void *), void *arg);*
- 새 스레드를 생성하고 시작합니다.
- *thread* : 해당 주소에 스래드의 id를 저장해 줍니다.
- *start_routine* : 스레드가 시작할 함수를 지정합니다. 즉, 새로운 스레드가 만들어지면 그 스레드는 *start_routine*이 가리키는 함수에서 시작하게 됩니다.
- *arg* : 스레드의 *start_routine*에 전달할 인자입니다.
- *return* : 스레드가 성공적으로 만들어졌으면 0, 에러가 있다면 0이 아닌 값을 반환합니다.

Specification - API (thread_exit)

- *void thread_exit(void *retval);*
- 스레드를 종료하고 값을 반환합니다. 모든 스레드는 반드시 이 함수를 통해 종료하고, 시작 함수의 끝에 도달하여 종료하는 경우는 고려하지 않습니다.
- *retval* : 스레드를 종료한 후 join 함수에서 받아갈 값입니다.

Specification - API (thread_join)

- *int thread_join(thread_t thread, void **retval);*
- 지정한 스레드가 종료되기를 기다리고, 스레드가 *thread_exit*을 통해 반환한 값을 받아옵니다. 스레드가 이미 종료되었다면 즉시 반환합니다.
- 스레드가 종료된 후, 스레드에 할당된 자원들 (페이지 테이블, 메모리, 스택 등)을 회수하고 정리해야 합니다.
- *thread* : join할 스레드의 id입니다.
- *retval* : 스레드가 반환한 값을 저장해 줍니다.
- *return* : 정상적으로 join했다면 0을, 그렇지 않다면 0이 아닌 값을 반환합니다.

Specification - xv6와의 상호작용

- 스레드는 xv6에서 프로세스와 비슷하게 취급되어야 합니다. 즉, 과제에 명시된 일부 기능을 제외하고는 기본적으로 프로세스처럼 동작합니다.

- 스케줄러는 두 번째 과제에서 만든 스케룰러들이 아닌, xv6에 기본적으로 들어 있는 RR 스케줄러를 사용하도록 합니다. 즉, 다른 프로세스들과 마찬가지로 스레드들도 스케줄러에 의해 RR로 선택됩니다.
  - git clone을 새로 받은 후, 과제를 진행하셔도 무방합니다.

Specification - 시스템 콜

- 스레드가 xv6에서 하나의 프로세스로서 올바르게 동작하려면 여러 시스템 콜들에 대한 대처가 되어야 합니다.

- 과제에서는 다음과 같은 시스템 콜들에 대해 스레드가 잘 동작하는지를 평가합니다. 동시에 여러 스레드에서 각자 시스템 콜을 호출하더라도 꼬이는 일이 없어야 합니다.

- *fork* : 스레드에서 *fork*가 호출되면 기존의 *fork* 루틴을 문제 없이 실행해야 합니다. 즉, 해당 스레드의 주소 공간의 내용을 복사하고 새로운 프로세스를 시작할 수 있어야 하고, *wait* 시스템 콜로 기다릴 수도 있어야 합니다.

- *exec* : *exec*가 실행되면 기존 프로세스의 모든 스레드들이 정리되어야 하며, 그 중 하나의 스레드에서 새로운 프로세스가 시작하고 나머지 스레드는 종료되어야 합니다.

Specification - 시스템 콜

- *sbrk* : *sbrk*는 프로세스에게 메모리를 할당하는 시스템 콜입니다. 여러 스레드가 동시에 메모리 할당을 요청하더라도 할당해주는 공간이 서로 겹치면 안 되며 (예를 들어, 두 스레드에서 동시에 각각 2개의 페이지에 해당하는 메모리 할당을 요청했다면 총 4개의 페이지가 할당되어야 합니다), 요청받은 크기만큼을 올바르게 할당할 수 있어야 합니다. *sbrk*에 의해 할당된 메모리는 프로세스 내의 모든 스레드가 공유 가능합니다.

- *kill* : 하나 이상의 스레드가 kill 되면 그 스레드가 속한 프로세스 내의 모든 스레드가 모두 정리되고 자원을 회수해야 합니다.

- *sleep* : 한 스레드가 *sleep*을 호출하면 그 스레드만 요청된 시간 동안 잠들어 있어야 합니다. 자고 있는 상태에서도 kill에 의해서 종료될 수 있어야 합니다.

- *pipe* : *pipe*는 각 스레드에서 각각 화면에 출력하는 데에 문제가 없도록만 합니다.

Locking

- p3_test 폴더의 “pthread_lock_linux.c” 파일을 수정하여 lock과 unlock을 구현해 보는 과제입니다.
- 해당 파일에 존재하는 “thread_func” 함수는 여러 쓰레드가 동시에 실행할 경우 race condition을 유발할 수 있습니다.
- 이를 해결하기 위한 lock과 unlock을 C 라이브러리에서 제공하는 동기화 API(ex: pthread_mutex)를 사용하지 않고 구현하세요.

Locking – common

- lock/unlock 함수의 매개변수와 함수의 몸통 부분을 자유롭게 추가하면 됩니다.
- 다른 변수나 lock/unlock 함수 이외의 함수를 정의하거나 및 사용하는 것은 상관없으나 pthread_mutex 같이 C library에 이미 존재하는 동기화 API는 사용해선 안됩니다.
- 컴파일 할 때는 맨 마지막에 –lpthread를 추가하면 됩니다.
  (ex: gcc -o pthread_lock_linux pthread_lock _linux.c -lpthread)
- NUM_ITERS의 값과 NUM_THREADS의 값을 수정하며 테스트 해보시길 바랍니다.

Locking – wiki

- 실제로 위의 조건을 만족하며 lock unlock을 구현할 수 있는지는 미지수이므로 위키 점수를 많이 반영할 계획입니다.
- 단, 테스트 케이스를 전부 통과한다면 locking 부분의 wiki를 적지 않아도 최고 점수를 받을 수 있으며, 아무리 wiki를 잘 쓰더라도 테스트 케이스를 전부 통과한 사람보다 높은 점수를 받을 수 없습니다.

- locking 부분 위키는 아래와 같은 내용을 포함해야 합니다.
  - lock/unlock 구현을 위한 아이디어(알고리즘 등).
  - 자신이 구현한 lock/unlock이 잘 작동하지 않을 경우 작동하지 않는 이유를 분석한 내용.

Test program

- 스레드의 기본적인 기능을 테스트 해볼 수 있는 테스트 프로그램이 제공됩니다.
- 이는 과제의 편의성을 위한 것으로, 제공되는 테스트 프로그램 이외의 테스트가 존재할 수 있습니다.
- 테스트는 명세를 넘어가는 기능을 요구하지 않으며, 몇몇 예외상황에 대한 테스트는 있을 수 있습니다.

참고사항

- 스레드는 프로세스와 비슷하므로, proc 구조체를 잘 다듬어 보시기 바랍니다.
- 프로세스의 페이지 테이블을 스레드 간에 공유하면 주소 공간을 공유할 수 있습니다.
- 어떤 부분을 어떻게 고쳐야 할지 막막하다면, 기존의 xv6 코드를 보고 많은 힌트를 얻을 수 있습니다. *fork, exec, exit, wait* 등의 함수가 어떤 식으로 구현되어 있는지를 분석해 보세요.
- 스레드 간 꼬이는 것을 방지하기 위해, 스레드 관련된 작업 중에는 ptable에 대한 lock을 고려하는 것이 좋습니다.
- 올바르게 구현된 스레드가 어떤 것인지를 확인하기 위해, 리눅스에서 pthread를 사용하여 다양한 케이스를 실행해보는 것을 추천드립니다.

Evaluation

- **Completeness** : 명세의 요구조건에 맞게 xv6가 올바르게 동작해야 합니다.
- **Defensiveness** : 발생할 수 있는 예외 상황에 대처할 수 있어야 합니다.
- **Wiki & Comment** : 테스트 프로그램과 위키를 기준으로 채점이 진행되므로, 위키는 최대한 상세히 작성되어야 합니다.
- **Deadline** : 데드라인을 반드시 지켜야 하며, 데드라인 이전 마지막 제출물을 기준으로 채점합니다.
- **DO NOT SHARE AND COPY !!**

Wiki

- **Design** : 스레드를 어떻게 구현할지, 추가/변경되는 컴포넌트 들이 어떻게 상호작용 하는지, 이를 위해 어떠한 자료구조들이 필요한지 등을 서술합니다.
- **Implement** : 실제 구현과정에서 변경하게 되는 코드영역이나 작성한 자료구조 등에 대하여 서술합니다.
- **Result** : 스레드가 올바르게 동작하고 있음을 확인할 수 있는 실행 결과와 이에 대한 설명을 서술합니다.
- **Trouble shooting** : 코드 작성 중 생겼던 문제와 이에 대한 해결 과정을 서술합니다.
- and whatever you want :)

Submission

- 구현한 코드와 wiki를 LMS 과제란에 제출합니다.
- WiKi, xv6-public 파일 및 locking 코드 전체를 한 디렉토리에 포함시킨 후, 압축하여 제출합니다.
- 제출할 디렉토리의 이름은 "OS_project03_수업번호_학번" 입니다.
- Wiki 파일의 이름은 "OS_project03_수업번호_학번.pdf" 입니다.
  - 제출할 디렉토리의 양식 예시는 다음과 같습니다.
    - ex) OS_project03_12345_2022123123

      -- xv6-public

      -- OS_project03_12345_2022123123.pdf

      -- locking source codes
    - 해당 디렉토리를 압축하여 OS_project03_12345_2022123123.zip 파일을 제출합니다.
- **제출 기한: 2024년 5월 19일 11:59 pm**
- **추가 제출 기한: 2024년 5월 20일 11:59 pm (Penalty of 50% of the total score)**

Thank you