Project

OS HW-2

모바일시스템공학과

32204288 조민욱

Contents

Introduction

Analyzing and planning

Producer 쓰레드와 Consumer 쓰레드는 각자 독립적으로 동작한다. 주어진 조건들을 활용하여 특정 작업을 수행하며, Mutex Lock 을 통해 다른 쓰레드 접근 방지와 허용을 한다. 쓰레드간 상호 의존적인 공유 오브젝트는 엄격하게 관리되어야 하며, 동작 중 Deadlock 과 Race condition 이 없는 효과적인 알고리즘을 짜야 한다. 즉 공유자원을 사용하지 않을 때는 즉각적으로 Mutex 반환을 해줘야 하며, 플래그 값에 따라 특정 쓰레드가 예상치 못하게 멈추는 일이 없도록 해야 한다.

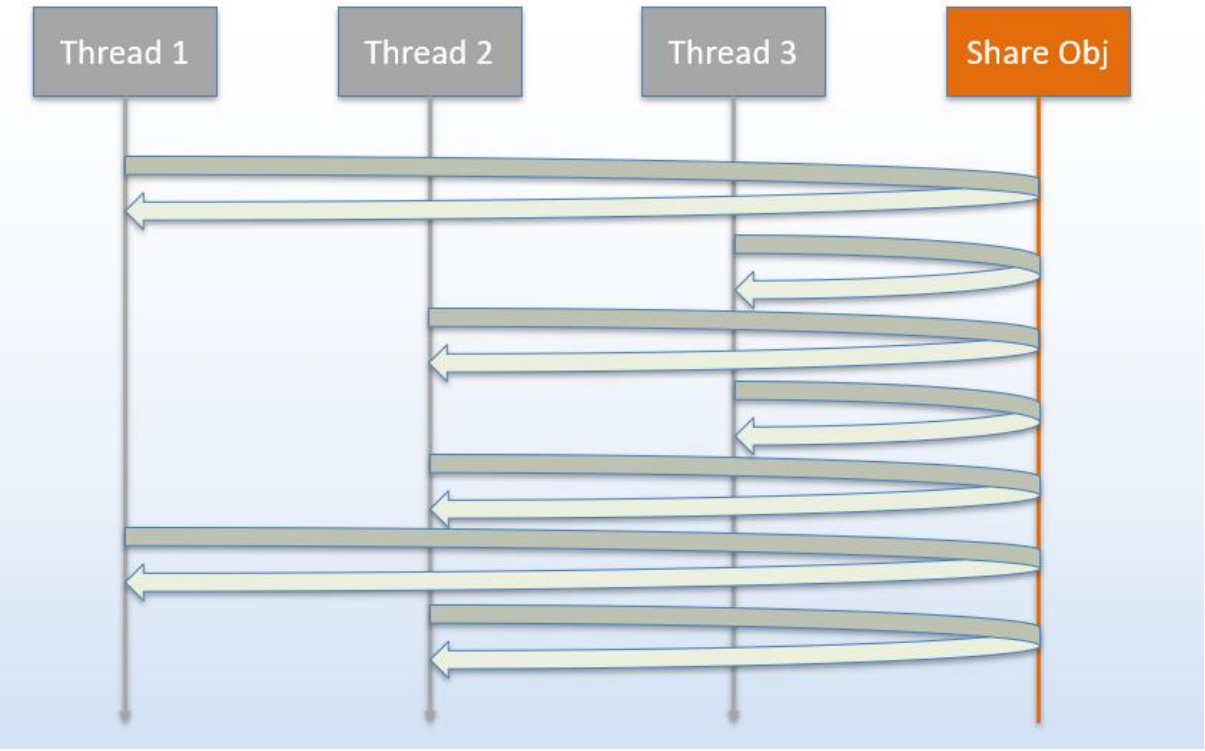

Figure 1) Simple Thread operation logic

최소한으로 빠르게 공유자원을 읽은 후, Unlock 을 통해 다른 쓰레드들이 바로 동작할 수 있도록 하면 된다. 이렇기에 같은 동작을 목표로 하더라도 실제 구현 코드와 아이디어는 천차만별이다. 기본적으로 깃 허브에 제공된 코드를 이용하며, 나만의 설계를 통해 완성하는 것을 목표로 했다.

Main explanation

Important concept

Prod 쓰레드는 파일로부터 읽은 라인이 없는(-1)인 경우와, Cons 쓰레드가 전달받아야 할 공유 오브젝트의 문자열이 비지 않은 경우를 다뤄야 한다.

```
while (1) {
    pthread_mutex_lock(&so->lock);

    // 기존 라인이 소비되지 않았을 경우 대기
    while (so->line != NULL) {
        pthread_mutex_unlock(&so->lock);
        usleep(100);
        pthread_mutex_lock(&so->lock);
    }

    read = getdelim(&line, &len, '\n', rfile);
    if (read == -1) {
        so->line = NULL;
        so->full = 1;
        pthread_mutex_unlock(&so->lock);
        break;
    }
```

Figure 2) Producer Thread – 1

1. Cons 가 라인을 소비하지 않은 경우 → **line != NULL**
2. End of line 을 만난 경우 → **read = -1**

반복 while 을 통해 쓰레드를 대기시키는 코드는 ChatGPT 도움을 받아 완성했다. 기존 라인을 NULL 로 지정하는 경우는 Prod 가 마지막 줄을 만났을 때와 Cons 가 해당 줄의 알파벳 수를 세려고 처리할 때일 뿐이다. 여기서 쓰레드를 대기시키는 코드를 앞에 배치함으로써 Cons 가 라인을 받기 전에 Prod 의 다른 쓰레드가 NULL 로 읽어야 할 라인을 없애는 경우를 방지했다. 어쨌든 위 과정을 모두 통과 시, 새 라인을 공유 구조체에 저장하고 Mutex unlock 과정을 거친다.

```
// 새 라인을 저장
so->linenum++;
so->line = strdup(line);
so->full = 1;

pthread_mutex_unlock(&so->lock);
i++;
```

Figure 3) Producer Thread – 2

Cons 쓰레드는 Prod 의 공유변수 플래그 값 지정에 맞춰 반응하면 된다. 단순하게 다음 두가지 과정을 필히 거치게 되고 플래그 값을 확인한다.

1. Prod 종료 신호를 확인한 경우 → **line = NULL & full = 1**
2. 읽은 line 이 빈 경우 → **line = NULL**

읽은 line 의 NULL 값 유무만 확인하게 된다면 Cons 쓰레드 두개가 연속으로 실행될 때 특정 쓰레드를 죽이게 된다. 즉 더 이상 작업할 게 없다는 시그널과 새로울 줄을 읽을 수 있다는 시그널을 분류하기 위해 full 변수를 적극 활용한다. 앞서 말한 케이스와 유사하게, 어떤 Cons 가 줄을 읽은 후 line=NULL & full=0 이 되기에, 바로 또 다른 Cons 쓰레드가 들어온다면 빈 줄을 처리하는 문제점이 생긴다. 따라서 구조체 라인을 확인 후

continue 를 통해 처음부터 다시 확인해야 한다. 이전 플래그 값이 매우 유동적이기에 이 경우 while 을 통해 쓰레드를 대기시키면 안 된다.

```
while (1) {
    pthread_mutex_lock(&so->lock);

    if (so->line == NULL && so->full == 1) {
        pthread_mutex_unlock(&so->lock);
        break;
    }

    line = so->line;
    so->line = NULL;
    so->full = 0;
    pthread_mutex_unlock(&so->lock);

    if (line == NULL) continue;
    else i++;

    char *line_copy = strdup(line);
    free(line);
```

Figure 4) Consumer Thread

이후 깃 허브에 참고된 char stat 코드를 이용하여 알파벳 수를 센다. 그동안 다른 쓰레드들은 열심히 자신의 작업을 한다. 처음에 빠르게 구조체 값을 읽어 Mutex unlock 을 해준 뒤, 독립적으로 자신이 할 일을 하고 조건에 맞지 않으면 다시 Mutex 대기를 하는 방식으로 효율적으로 동작한다.

Unique features

앞서 Mutex 를 이용한 동기화 외에도, Thread condition 조건 변수를 활용한 방식이 있다. Mutex lock 과 Mutex unlock 사이에 wait 과 broadcast 함수를 적용시켜 Prod 와 Cons 에 각각 신호를 보내게 된다.

```c
while (1) {
    pthread_mutex_lock(&so->lock);

    while (so->line != NULL) pthread_cond_wait(&so->cond_prod, &so->lock);

    read = getdelim(&line, &len, '\n', rfile);
    if (read == -1) {
        so->full = 1;
        pthread_cond_broadcast(&so->cond_cons);
        pthread_mutex_unlock(&so->lock);
        break;
    }
    so->linenum = i;
    so->line = strdup(line);
    so->full = 1;
    i++;

    pthread_cond_broadcast(&so->cond_cons);
    pthread_mutex_unlock(&so->lock);
}
```

Figure 5) Producer Thread with condition variable

```c
while (1) {
    pthread_mutex_lock(&so->lock);

    while (!so->full) pthread_cond_wait(&so->cond_cons, &so->lock);

    line = so->line;
    if (line == NULL && so->full == 1) {
        pthread_cond_broadcast(&so->cond_prod);
        pthread_mutex_unlock(&so->lock);
        break;
    }
    len = strlen(line);

    so->line = NULL;
    so->full = 0;
    pthread_cond_broadcast(&so->cond_prod);
    pthread_mutex_unlock(&so->lock);

    char *line_copy = strdup(line);
    free(line);
    i++;
```

Figure 6) Consumer Thread with condition variable

다만 쓰레드 condition 조건 변수를 활용한 test2 파일의 경우 올바른 값을 출력하나, 쓰레드가 늘어날수록 오히려 큰 시간을 잡아먹는다.

Outro

Environment

Linux environment. Basically, ASSAM server and Local Ubuntu.

- Test1 → Runs more effectively on Local Ubuntu
- Test2 → Runs more effectively on ASSAM server

Screenshots

```
adrd@ADALIV:~/2024-os-hw2$ time ./a.out FreeBSD9-orig.tar 100 100
Final Alphabet Count:
      A       B       C       D       E       F       G       H       I       J       K       L       M
14239598  4972835 11303608  9549671 22447966  7962974  3715946  4591418 14220229   344062  1483389  8150586  5628639
      N       O       P       Q       R       S       T       U       V       W       X       Y       Z
12350324 10792819  6635469   512006 12521644 13178317 16491358  6070339  2637484  1777558  7532456  2215212   510983

real    2m16.907s
user    2m48.017s
sys     31m37.429s
```

Figure 7) test1 with multi threads (Huge file input)

```
adrd@ADALIV:~/2024-os-hw2/src$ time ./a.out test 1 1
```

```
Final Alphabet Count:
      A       B       C       D       E       F       G       H       I       J       K       L       M
  15456   15456   15456   15456   15456   15456   15456   15456   15456   15456   15456   15456   15456
      N       O       P       Q       R       S       T       U       V       W       X       Y       Z
  15456   15456   15456   15456   15456   15456   15456   15456   15456   15456   15456   15456   15456

real    0m27.611s
user    0m27.382s
sys     0m0.686s
```

Figure 8) test1 with 1 Producer and 1 Consumer

```
adrd@ADALIV:~/2024-os-hw2/src$ time ./a.out test 100 1
```

```
Final Alphabet Count:
      A       B       C       D       E       F       G       H       I       J       K       L       M
  15456   15456   15456   15456   15456   15456   15456   15456   15456   15456   15456   15456   15456
      N       O       P       Q       R       S       T       U       V       W       X       Y       Z
  15456   15456   15456   15456   15456   15456   15456   15456   15456   15456   15456   15456   15456

real    0m5.873s
user    0m6.896s
sys     0m22.436s
```

Figure 9) test1 with 100 Producer and 1 Consumer

```
adrd@ADALIV:~/2024-os-hw2/src$ time ./a.out test 1 100
```

```
Final Alphabet Count:
      A       B       C       D       E       F       G       H       I       J       K       L       M
  15456   15456   15456   15456   15456   15456   15456   15456   15456   15456   15456   15456   15456
      N       O       P       Q       R       S       T       U       V       W       X       Y       Z
  15456   15456   15456   15456   15456   15456   15456   15456   15456   15456   15456   15456   15456

real    0m16.973s
user    0m20.139s
sys     4m9.604s
```

Figure 10) test1 with 1 Producer and 100 Consumer

```
adrd@ADALIV:~/2024-os-hw2/src$ time ./a.out test 100 100
```

```
Final Alphabet Count:
      A       B       C       D       E       F       G       H       I       J       K       L       M
  15456   15456   15456   15456   15456   15456   15456   15456   15456   15456   15456   15456   15456
      N       O       P       Q       R       S       T       U       V       W       X       Y       Z
  15456   15456   15456   15456   15456   15456   15456   15456   15456   15456   15456   15456   15456

real    0m0.665s
user    0m0.720s
sys     0m8.936s
```

Figure 11) test1 with 100 Producer and 100 Consumer

```
solid@solid-00001-58989694-7jl4n:~/2024-os-hw2$ time ./a.out  FreeBSD9-orig.tar 1 1
Cons_10b9640: 21262918 lines
Prod_18ba640: 21262918 lines

Final Alphabet Count:
      A       B       C       D       E       F       G       H       I       J       K       L       M
14239598  4972835 11303608  9549671 22447966  7962974  3715946  4591418 14220229   344062  1483389  8150586  5628639
      N       O       P       Q       R       S       T       U       V       W       X       Y       Z
12350324 10792819  6635469   512006 12521644 13178317 16491358  6070339  2637484  1777558  7532456  2215212   510983

real    2m17.184s
user    0m45.269s
sys     1m44.162s
```

Figure 12) test2 with 1 Producer and 1 Consumer

```
solid@solid-00001-58989694-7jl4n:~/2024-os-hw2$ time ./a.out  FreeBSD9-orig.tar 3 3
Cons_6ffff640: 7146255 lines
Prod_754c8640: 7078071 lines
Cons_6f7fe640: 7012974 lines
Prod_75cc9640: 7168159 lines
Cons_74cc7640: 7103689 lines
Prod_764ca640: 7016688 lines

Final Alphabet Count:
      A       B       C       D       E       F       G       H       I       J       K       L       M
14239598  4972835 11303608  9549671 22447966  7962974  3715946  4591418 14220229   344062  1483389  8150586  5628639
      N       O       P       Q       R       S       T       U       V       W       X       Y       Z
12350324 10792819  6635469   512006 12521644 13178317 16491358  6070339  2637484  1777558  7532456  2215212   510983

real    6m47.372s
user    3m47.531s
sys     9m21.701s
```

Figure 13) test2 with 3 Producer and 3 Consumer

```
solid@solid-00001-58989694-7jl4n:~/2024-os-hw2$ time ./a.out FreeBSD9-orig.tar 50 50
```

```
Final Alphabet Count:
       A       B       C       D       E       F       G       H       I       J       K       L       M
14239598  4972835 11303608  9549671 22447966  7962974  3715946  4591418 14220229   344062  1483389  8150586  5628639
       N       O       P       Q       R       S       T       U       V       W       X       Y       Z
12350324 10792819  6635469   512006 12521644 13178317 16491358  6070339  2637484  1777558  7532456  2215212   510983

real    130m13.116s
user    20m44.773s
sys     234m28.080s
```

Figure 14) test2 with 50 Producer and 50 Consumer