Multi Thread

Homework 2
모바일시스템공학과
32143153 이대훈

1. Project Introduction

Process는 현재 실행중인 Program을 의미합니다. 이러한 Process에서 Thread는 process 내에서 실행되는 최소 단위를 말합니다. Thread는 Process내에 존재하는 데이터를 공유합니다. 이러한 점을 보완하기위해 Mutual exclusion을 보장해줍니다. MultiThread는 프로그램이 실행할 때 Thread끼리 서로의 영향을 받지 않고 동시에 돌 수 있는 작업을 의미합니다.

2. Motivation

프로그램을 실행할 때 현재실행 중인 프로그램을 프로세스라고 합니다. 이러한 프로세스 안에서 동작하는 최소단위를 스레드라고 합니다. 단순히 스레드가 동작하는 것이 아닌 OS에서 정해준 시간에 대해서 스레드가 동작하고 schedule out을 합니다. 하지만 이러한 과정에서 schedule out이 언제 될지를 모르기 때문에 mutual exclusion이 broken 됩니다. 이러한 점을 보장해주기 위해 synchronization을 해줍니다. 이러한 점에 대해 공부하며 실제로 컴퓨터 내에서 프로세스의 동작과정을 이해할 수 있습니다.

3. Instruction

3-1) Structure 선언

so_t와 buffer라는 두 개의 structure를 만듭니다. buffer는 so_t structure 안에 배열로 다시 입력해 줍니다. 이를 통해 main에서 structure를 생성하고 so_t structure를 공유하는 스레드들이 생성됩니다. 이 때 Multithread를 하기위해 buffer는 각각의 스레드들에 해당하는 값들을 입력해줍니다. Multithread를 위해 buffer를 통해서 synchronization을 합니다.

3-2) Producer 함수

Producer 함수는 공유 데이터의 buffer안에 문자열을 넣어주는 역할을 합니다. Producer는 처음에 있는 buffer[0]부터 채워나가기 시작합니다. 처음 버퍼에 대해서 문자열을 채우면 다음 번 buffer에 문자열을 입력합니다. 이 때 producer가 접근해 있는 buffer에 consumer가 접근하지 못하게 하기위해서 접근하고 있는 buffer에 대해서 mutex_lock을 합니다. 이를 통해 producer가 buffer에 접근하면 consumer는 producer가 critical section을 종료하고 mutex_lock을 해제해줄 때까지 접근할 수 없습니다. 이를 통해 mutual exclusion을 보장하고 buffer에 대해 synchronization을 할 수 있습니다. Producer에서 선택한 buffer가 비었는지와 차있는지를 구분하기위해 full로 나타냅니다. 차있다면 full = 1, 비었다면 full = 0을 가지고 있습니다. buffer가 비어있다면 producer는 buffer에 문자열을 입력하면 됩니다. 하지만 buffer가 차있다면 producer가 문자열을 입력할 경우 그전에 가지고 있던 문자열은 지

워지고 새로운 문자열이 buffer에 들어갑니다. 이러한 점 때문에 buffer가 차있다면 안에 있는 문자열이 소모될 때까지 기다려주는 작업이 필요합니다. 이 때 full == 1이라는 조건을 지속적으로 확인해주는 것을 spinlock이라하며 이러한 경우에 CPU를 많이 사용합니다. 이러한 점을 방지하고자 wait을 사용합니다. wait을 하게 되면 signal이 오기 전까지는 잠이 듭니다. 다시 말해 consumer에서 buffer를 소모하고 signal을 보내주어 wait 상태를 깨워준다면 이 때 full = 0이 되고 producer는 문자열을 해당 buffer에 입력해주면 됩니다. wait 상태에서는 signal이 오기 전까지 동작하지 않으므로 지속적인 확인이 필요 없습니다. 이러한 부분에서 CPU소모를 줄일 수 있습니다. 이후 producer를 실행하다가 넣어주는 문자열이 더 이상 없게 되면 producer thread를 종료시킵니다.

3-3) Consumer

Consumer 또한 producer와 큰 차이가 없습니다. 하지만 이 때 Multi consumer를 사용하기 위해 다음과 같이 3가지 방법을 사용해 보았습니다.

1. Buffer와 consumer thread간의 1:1 연결

Buffer와 consumer를 1:1 연결을 위해 각각의 consumer가 생성될 때 초기 값으로 buffer의 index를 지정해줍니다. 이렇게 함으로써 consumer가 while문 안에서 지속적으로 실행될 때 index는 하나만 사용되어 지속적으로 하나의 buffer만을 바라보고 있게 됩니다. 여기서 consumer의 index를 지정해주는 것에 대해서도 mutual exclusion을 보장해주어야 합니다. Consumer의 index를 받아 올 때 synchronization을 해주지 않는다면 같은 index를 갖는 consumer thread가 생성됩니다. 여기서 mutex_lock에 대한 변수도 생각해 줄 수 있습니다. 변수로 buffer와 같은 변수를 사용한다고 하면 producer thread와 consumer thread 둘이 경쟁하던 것이 초기 index 값 선언을 위해 3개가 경쟁하는 형태로 변환됩니다, 이러한 점을 해결해주기 위해 새로운 변수를 설정하여 consumer에서 초기 index 정하는 부분에 대해 consumer들끼리만 경쟁하도록 해줍니다. 이렇게 1:1 연결이 완료되면 producer와 마찬가지로 consumer가 buffer에 접근할 때 producer가 같은 buffer에 들어오지 못하도록 lock을 설정합니다. 처음 buffer를 확인하기 전 producer가 끝났는지 끝나지 않았는지에 확인을 해줍니다. Producer가 끝난 이후에 buffer가 비어있다면 이 consumer는 더 이상 할 작업이 없기 때문에 종료해도 됩니다. producer가 끝나기 전이라면 buffer가 차있는지 비었는지를 확인해주고 차있으면 빼내고 없다면 wait을 하여 buffer안에 문자열이 들어오고 신호가 와서 깨워줄 때까지 기다립니다. 이후에는 producer와 비슷한 작업들을 해줍니다. 이 때 consumer는 종료하는 과정에서 signal을 보내는 위치가 다릅니다. consumer가 종료한다는 것은 producer가 끝났다는 것이고 가리키고 있는 buffer에 대해서 더 이상 signal을 보내줄 필요가 없습니다. 하지만 다른 consumer thread에 대해서 signal들을 보내주어야 합니다. 다른 consumer thread에서는 wait 상태에 들어가 있는 경우가 있기 때문입니다. 이러한 문제를 해결해주기 위해 consumer가 종료되면서 다음 buffer에 대해 signal을 보내줍니다.

2. Buffer와 consumer thread가 1:1 연결이 아닌 경우

위에서 consumer의 index 초기 값을 미리 정해 각각의 consumer thread 마다 index를 고정하였다면 1:1연결이 아닌 경우에 대해서는 consumer thread가 실행할 때마다 index값을 받아오게 합니다. 이 때도 마찬가지로 mutual exclusion을 사용하여 index가 겹치지 않게

해주었습니다.

3. Buffer와 consumer thread가 1:1 연결이 아니고 wait을 하지 않는 경우

다음은 위와는 다르게 wait을 뺀 경우의 코드입니다. wait을 빼고 조건을 확인했을 때 full이 0이면 continue를 사용하여 맨 처음으로 돌아가게 합니다. 이 때 다시 받아오는 index는 1 증가한 값으로 다음번에 조건이 맞을 때까지 실행 순서를 미룰 수 있습니다. 이렇게 하면 조건을 만족하는지에 대해 그 때 마다 확인을 해야 한다는 점에서 CPU의 사용량이 많아지지만 많은 buffer들이 wait하는 경우에 비해 속도를 높일 수 있었습니다.

3-4) char_stat

char_stat 함수를 통해 line에 들어있는 문자열들을 구분하고 각각의 문자에 맞게 stat array에 저장해주는 작업을 합니다. 여기서 stat은 consumer에서 실행하는 함수이며 parameter로 line을 받아옵니다. 이 때 char_stat 자체에 대해서 mutual exclusion을 해주어야 합니다. stat array는 전역변수로 선언한 배열입니다. 각각의 consumer에서 전역변수에 접근할 때 공유 데이터처럼 서로가 동시에 사용할 수 있습니다. 이러한 경우를 방지해주기 위해 consumer에서 사용할 때 새로운 변수를 사용하여 mutex_lock해주었습니다.

3-5) main

main에서는 각각의 structure 및 값들에 대해서 초기화를 해줍니다. 특히 처음 사용하는 mutex_lock에 대해서 pthread_mutex_init으로 초기화합니다. 변수로는 초기화하는 변수와 다음으로는 초기화한 변수의 기능을 정해주는 것이라고 합니다. NULL을 사용할 경우 가장 기본적인 기능으로 초기화를 합니다. pthread_create을 사용하여 각각의 thread들을 생성하고 마지막에 pthread_join을 통해 생성한 thread들을 삭제합니다. 마자막으로 stat array에 저장된 각각의 문자를 출력합니다.

4. Program Structure

1. Structure & Global value initialize

2. Producer
    2-1. Local value initialize
    2-2. Loop

    2-3. Empty?
    2-4. File done then break
    2-5. Put the line to buffer
    2-6. Increase index for next buffer

3. Consumer
    3-1. Local value initialize
    3-2. Loop

3-3. Increase index after put current index
3-4. producer break?
3-5. Full?
3-6. Producer all done && nothing in buffer then break
3-7. Take the line from buffer

5. Build environment

Compilation : Linux Assam, with GCC
To compile : gcc Multicon1.c -lpthread, gcc Multicons2.c -lpthread. gcc Multicons2.c -lpthread
To run : time ./a.out Filename number of producer number of consumer
input file : FreeBSD9-orig.tar

6. Problems and solutions

6-1) Multi thread에 대한 이해를 잘못했었습니다. Multi thread라는 것이 단순히 여러 개의 thread가 순차적으로 진행한다고 생각하였습니다. 하지만 Multi thread는 동시에 여러 개의 thread가 진행하도록 하는 것이었습니다. 이러한 부분을 해결하기 위해 각각 thread 전체에 걸려 있는 lock을 각각 buffer에 대해서 lock을 걸어주도록 개선하였습니다. Buffer를 통해 lock을 걸면 그 buffer에 대해서만 lock이 걸리므로 다른 buffer를 사용하는 thread들은 동시에 함께 진행할 수 있습니다.
6-2) 지역변수와 전역변수에 대해서 다시 한 번 생각해 볼 수 있었습니다. Thread는 서로의 데이터를 공유할 수 있습니다. 이럴 때 단순히 Structure로 만들어진 변수들만 공유하는 것이 아닌 전역변수도 서로 공유하는 자원입니다. 그렇기 때문에 전역변수를 설정해놓았다면 각각 사용함에 있어서 mutual exclusion을 보장해주어야 합니다. 반면 지역변수는 각각의 thread 내에서 선언한 변수이므로 공유하는 자원이 아닙니다. 이를 통해 두 변수의 차이에 맞게 작업을 해주었습니다.
6-3) Critical Section을 줄이면 그만큼 thread가 동시에 진행 할 수 있는 영역이 많아집니다. 같은 lock변수를 사용하는 producer와 consumer에 대해서 공유 데이터가 아닌 각자의 데이터 즉 mutual exclusion을 보장해 줄 필요 없는 변수를 사용할 때 lock 밖에서 또는 lock을 풀고 다시 걸어주는 과정을 통해 critical section을 줄일 수 있었습니다.
6-4) free()를 사용함에 있어서 error가 발생했습니다. free를 사용할 때 double free or corruption(fasttop)이라는 error가 나왔고 이를 해결해주기 위해 free한 변수를 NULL로 다시 입력해주었습니다. 다시 입력을 해주지 않을 경우에 대해 memory leak이 발생한다고 합니다. 이 와 마찬가지로 strdup()은 문자열길이에 맞게 저장 공간을 동적 할당하는 함수지만 동적 할당을 활용 한만큼 free를 사용해 주어야 합니다.
6-5) Signal을 보낼 때 다양한 경우에 대해서 생각하는 부분이 많은 시간을 차지했습니다. Signal을 보내면 signal을 받은 wait은 깨워지고 다시 한 번 조건을 확인하게 됩니다. 이 때 내가 깨우고자 하는 wait을 직접 나타내기 위해 cond 변수 또한 buffer structure 안에 집어넣었습니다. 이를 통해 buffer의 index에 따라 cond 변수도 서로 구분을 할 수 있습니다. 특히 consumer가 break할 때에 대해 signal을 보내주는 곳의 위치가 중요했습니다.

consumer가 break 된다는 것은 이미 producer가 종료되었다는 의미이므로 consumer가 다시 같은 buffer index에 대해 signal을 보낼 필요가 없습니다. 하지만 signal을 보내지 않고 consumer가 종료한다면 다른 consumer들은 signal을 받을 상대가 없습니다. 이러한 경우에 wait이 계속이어지면서 deadlock이 발생합니다. 이러한 문제를 해결해주기 위해 consumer가 break할 때 signal은 각각 다음 buffer index에 cond에 보내게 하였습니다. 또한 이런 경우도 있습니다. Signal을 보냈는데 보낸 당시에는 wait을 하지 않고 있어 문제가 없고 Signal이 보내진 이후에 wait이 걸리면 이젠 어디서 Signal을 받을 수 있을지에 대해서 고민했습니다. 이러한 경우에는 조건을 추가하여 producer가 끝난 이후에 대해 buffer가 비어있는 경우 앞으로 들어올 문자열이 없기 때문에 wait을 하지 못하고 지나가도록 하였습니다. 이처럼 Signal이 보내질 때 같은 변수 cond가 여러 곳에서 wait이 되어 있으면 어떤 wait에 이 신호가 갈지 알 수 없습니다. Signal은 한 번에 하나의 wait만을 깨우기 때문입니다. 동시에 같은 변수로 잠들어 있는 wait을 모두 깨우는 방법으로는 broadcast가 있다고 합니다.

6-6) 시간을 얼마나 많이 소모하는지에 대해서 문제를 해결하기 위해 buffer와 consumer의 1:1 연결에서 consumer가 무작위로 buffer를 나타내도록 하였습니다. buffer와 consumer가 1:1일 경우 consumer와 buffer의 개수가 같은 만큼에서만 효율이 증가합니다. buffer가 consumer thread의 개수보다 많은 경우에 대해서 남은 buffer는 사용하지 못하는 자원입니다. 하지만 이러한 부분에 대해 1:1 연결이 아니라면 남은 buffer들 또한 사용이 가능해집니다.

6-7) 다른 방법으로 시간적인 효율을 확인해 보기위해 Multi producer와 Multi consumer 모두를 각각의 buffer에 대해 1:1:1 연결을 시도해 보았습니다. 이러한 경우에는 producer와 consumer의 개수가 서로 같아야합니다. 하지만 이 부분에 대해서 생각 보다 측정한 값은 오랜 시간이 걸렸습니다. Multi producer를 효율적으로 사용하기 위해서는 받아오는 file을 producer의 수만큼 쪼개어 각자 다른 부분을 buffer에 넣어주는 방식으로 진행을 하여야했습니다. 하지만 이 부분에 있어서는 실제로 진행해 보지 못하였습니다.

6-8) Buffer의 개수를 무한히 할 수 없기 때문에 circular queue를 사용하여 만들었습니다. Buffer 개수를 초과하는 index가 생성되면 buffer의 개수로 나누어 나머지를 index로 만들어 줍니다. 이를 통해 유한한 buffer로 지속적인 작업을 할 수 있습니다.

7. Personal feelings

이번 Multi thread를 진행하면서 참 많은 어려움을 느꼈습니다. 처음 Code를 만들 때 Multi thread라는 것이 단순히 순차적으로 thread가 진행한다고 생각하여 한 번에 하나의 thread가 실행하도록 구성하였습니다. 하지만 Multi thread는 순차적인 진행이 아닌 여러 thread가 한 번에 도는 것을 뜻했습니다. 이러한 문제점을 개선하기 위해 critical section을 정해주는 것에 대해 다시 알게 되었고 synchronization을 통한 mutual exclusion도 알게 되었습니다. 그 전과 달라진 부분은 코드를 컴파일 했을 경우에 error에 대한 내용이 없다는 점입니다. 그 전까지의 과제에서는 segmentation fault라고 error가 나오면 어디서 error가 발생했는지 알 수 있지만 deadlock이 발생하면 어떠한 경우에 그러한 문제가 발생했는지를 직접 생각하고 각각의 경우들을 다 확인해주는 작업이 필요했습니다. 이러한 점이 이번과제를 하면서 가장 힘들었던 부분이었습니다.

8. Code

<Multicon1.c> - Multi consumer와 buffer간에 1:1 연결
time ./a.out FreeBSD9-orig.tar 1 2 실행할 경우 : 1m54.443s

```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <time.h>
#define ASCII_SIZE        256
int stat [ASCII_SIZE];
void char_stat(char* argv);
int Ncons;
pthread_mutex_t STAT;
pthread_mutex_t con;
int gc;

typedef struct buffer {
        char *line;
        int full;
        int linenum;
        int count;
        pthread_mutex_t lock;
        pthread_cond_t cond;
}buffer;

typedef struct sharedobject {
        FILE *rfile;
        buffer buf[100];
        int flag;
        int out;
}so_t;

void *producer(void *arg) {
        so_t *so = arg;
        int *ret = malloc(sizeof(int));
        FILE *rfile = so->rfile;
        int i = 0;
        int p = 0;
        int temp;
```

```
        char *line = NULL;
        size_t len = 0;
        ssize_t read = 0;

        while (1) {
                read = getdelim(&line, &len, '\n', rfile);
                pthread_mutex_lock(&so->buf[p].lock);
                while(so->buf[p].full == 1)
                {
                        pthread_cond_wait(&so->buf[p].cond, &so->buf[p].lock);
                }
                if (read == -1)
                {
                        so->buf[p].line = NULL;
                        so->buf[p].full = 1;
                        so->flag = 1;
                        pthread_cond_signal(&so->buf[p].cond);
                        pthread_mutex_unlock(&so->buf[p].lock);
                        break;
                }
                so->buf[p].linenum = i;
                so->buf[p].line = strdup(line);        /* share the line */
                so->buf[p].full = 1;
                pthread_cond_signal(&so->buf[p].cond);
                pthread_mutex_unlock(&so->buf[p].lock);
                i++;
                p = (p + 1) % Ncons;
        }
        free(line);
        line = NULL;
        //printf("Prod_%x: %d lines\n", (unsigned int)pthread_self(), i);
        *ret = i;
        pthread_exit(ret);
}

void *consumer(void *arg) {
        so_t *so = arg;
        int *ret = malloc(sizeof(int));
        int i = 0;
        int len;
        char *line;
```

```
	int temp;
	pthread_mutex_lock(&con);
	int c = gc % Ncons;
	gc ++;
	pthread_mutex_unlock(&con);

	while (1){
		pthread_mutex_lock(&so->buf[c].lock);
		if(so->flag == 0)
		{
			while(so->buf[c].full == 0)
			{
				pthread_cond_wait(&so->buf[c].cond,
&so->buf[c].lock);
				if(so->flag == 1)break;
			}
		}
		line = so->buf[c].line;
		pthread_mutex_unlock(&so->buf[c].lock);
		if (line == NULL){
			pthread_mutex_lock(&so->buf[c].lock);
			temp = (c + 1) % Ncons;
			pthread_cond_signal(&so->buf[temp].cond);
			pthread_mutex_unlock(&so->buf[c].lock);
			break;
		}
		pthread_mutex_lock(&STAT);
		char_stat(line);
		pthread_mutex_unlock(&STAT);
		//printf("Cons_%x: [%02d:%02d] %s",
		//		(unsigned int)pthread_self(), i, so->buf[c].linenum,
so->buf[c].line);
		i++;
		pthread_mutex_lock(&so->buf[c].lock);
		free(so->buf[c].line);
		so->buf[c].line = NULL;
		so->buf[c].full = 0;
		pthread_cond_signal(&so->buf[c].cond);
		pthread_mutex_unlock(&so->buf[c].lock);
	}
	//printf("Cons: %d lines\n", i);
```

```
        *ret = i;
        pthread_exit(ret);
}

void char_stat(char *argv)
{
        size_t length = 0;
        char *cptr = NULL;
        char *substr = NULL;
        char *brka = NULL;
        char *sep = "{}()[],;\" \n\t^";
        cptr = strdup(argv);
        for (substr = strtok_r(cptr, sep, &brka); substr; substr = strtok_r(NULL,
sep, &brka))
        {
                length = strlen(substr);
                for (int i = 0; i < length; i++)
                {
                        if (*substr < 256 && *substr > 1)
                        {
                                stat[*substr]++;
                        }
                substr++;
                }
        }
        free(cptr);
        return;
}

int main (int argc, char *argv[])
{
        pthread_t prod[100];
        pthread_t cons[100];
        int Nprod;
        int rc;    long t;
        int *ret;
        int i;
        FILE *rfile;
        if (argc == 1) {
                printf("usage: ./prod_cons <readfile> #Producer #Consumer\n");
                exit (0);
```

```
}
buffer *buf = malloc(sizeof(buffer));
memset(buf, 0, sizeof(buffer));
so_t *share = malloc(sizeof(so_t));
memset(share, 0, sizeof(so_t));
// initialize stat
memset(stat, 0, sizeof(stat));
rfile = fopen((char *) argv[1], "r");
if (rfile == NULL) {
        perror("rfile");
        exit(0);
}
if (argv[2] != NULL) {
        Nprod = atoi(argv[2]);
        if (Nprod > 100) Nprod = 100;
        if (Nprod == 0) Nprod = 1;
} else Nprod = 1;
if (argv[3] != NULL) {
        Ncons = atoi(argv[3]);
        if (Ncons > 100) Ncons = 100;
        if (Ncons == 0) Ncons = 1;
} else Ncons = 1;

share->rfile = rfile;

for(i=0;i<Ncons;i++){
share->buf[i].line = NULL;
pthread_mutex_init(&share->buf[i].lock, NULL);
pthread_cond_init(&share->buf[i].cond, NULL);
}
pthread_mutex_init(&STAT, NULL);
pthread_mutex_init(&con, NULL);
for (i = 0 ; i < Nprod ; i++)
        pthread_create(&prod[i], NULL, producer, share);
for (i = 0 ; i < Ncons ; i++)
        pthread_create(&cons[i], NULL, consumer, share);
printf("main continuing\n");

for (i = 0 ; i < Ncons ; i++) {
        rc = pthread_join(cons[i], (void **) &ret);
        printf("main: consumer_%d joined with %d\n", i, *ret);
```

```
        }
        for (i = 0 ; i < Nprod ; i++) {
                rc = pthread_join(prod[i], (void **) &ret);
                printf("main: producer_%d joined with %d\n", i, *ret);
        }
        printf(" A : %8d\nB : %8d\nC : %8d\nD : %8d\nE : %8d\nF : %8d\nG :
%8d\nH : %8d\nI : %8d\nJ : %8d\nK : %8d\nL : %8d\nM : %8d\nN : %8d\nO :
%8d\nP : %8d\nQ : %8d\nR : %8d\nS : %8d\nT : %8d\nU : %8d\nV : %8d\nW :
%8d\nX : %8d\nY : %8d\nZ : %8d\n",
                        stat['A']+stat['a'],    stat['B']+stat['b'],        stat['C']+stat['c'],
stat['D']+stat['d'],  stat['E']+stat['e'],
                        stat['F']+stat['f'],    stat['G']+stat['g'],        stat['H']+stat['h'],
stat['I']+stat['i'],  stat['J']+stat['j'],
                        stat['K']+stat['k'],    stat['L']+stat['l'],        stat['M']+stat['m'],
stat['N']+stat['n'],  stat['O']+stat['o'],
                        stat['P']+stat['p'],    stat['Q']+stat['q'],        stat['R']+stat['r'],
stat['S']+stat['s'],  stat['T']+stat['t'],
                        stat['U']+stat['u'],    stat['V']+stat['v'],        stat['W']+stat['w'],
stat['X']+stat['x'],  stat['Y']+stat['y'],
                        stat['Z']+stat['z']);
        pthread_exit(NULL);
        exit(0);
}
```

<Multicons2.c> - Multi consumer가 buffer를 접근(1:1이 아닌), wait 대신 continue 사용
time ./a.out FreeBSD9-orig.tar 1 2 실행할 경우 : 0m24.447s

```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <time.h>
#define ASCII_SIZE        256
int stat [ASCII_SIZE];
void char_stat(char* argv);
pthread_mutex_t STAT;
pthread_mutex_t con;
int gc;

typedef struct buffer {
        char *line;
        int full;
        int linenum;
        pthread_mutex_t lock;
        pthread_cond_t cond;
}buffer;

typedef struct sharedobject {
        FILE *rfile;
        buffer buf[100];
        int flag;
}so_t;

void *producer(void *arg) {
        so_t *so = arg;
        int *ret = malloc(sizeof(int));
        FILE *rfile = so->rfile;
        int i = 0;
        int p = 0;
        int temp;
        char *line = NULL;
        size_t len = 0;
        ssize_t read = 0;
```

```
        while (1) {
                pthread_mutex_lock(&so->buf[p].lock);
                if(so->buf[p].full == 0)read = getdelim(&line, &len, '\n', rfile);
                if(so->buf[p].full == 1)
                {
                        temp = p;
                        p = (p + 1) % 100;
                        pthread_mutex_unlock(&so->buf[temp].lock);
                        continue;
                }
                if (read == -1)
                {
                        so->buf[p].line = NULL;
                        so->buf[p].full = 1;
                        so->flag = 1;
                        pthread_mutex_unlock(&so->buf[p].lock);
                        break;
                }
                so->buf[p].linenum = i;
                so->buf[p].line = strdup(line);        /* share the line */
                so->buf[p].full = 1;
                //i++;
                //temp = p;
                //p = (p + 1) % 100;
                pthread_mutex_unlock(&so->buf[p].lock);
                i++;
                p = (p + 1) % 100;
        }
        free(line);
        line = NULL;
        printf("Prod_%x: %d lines\n", (unsigned int)pthread_self(), i);
        *ret = i;
        pthread_exit(ret);
}

void *consumer(void *arg) {
        so_t *so = arg;
        int *ret = malloc(sizeof(int));
        int i = 0;
        int len;
        char *line;
```

```
	int temp;
	int c = 0;

	while (1){
		pthread_mutex_lock(&con);
		c = gc;
		gc = (gc + 1) % 100;
		pthread_mutex_unlock(&con);
		pthread_mutex_lock(&so->buf[c].lock);
		if(so->flag == 0)
		{
			if(so->buf[c].full == 0)
			{
				pthread_mutex_unlock(&so->buf[c].lock);
				continue;
			}
		}
		line = so->buf[c].line;
		if (line == NULL)
		{
			pthread_mutex_unlock(&so->buf[c].lock);
			break;
		}
		pthread_mutex_lock(&STAT);
		char_stat(line);
		pthread_mutex_unlock(&STAT);
		//printf("Cons_%x: [%02d:%02d] %s",
		//		(unsigned int)pthread_self(), i, so->buf[c].linenum,
so->buf[c].line);
		free(so->buf[c].line);
		so->buf[c].line = NULL;
		so->buf[c].full = 0;
		//i++;
		pthread_mutex_unlock(&so->buf[c].lock);
		i++;
	}
	printf("Cons: %d lines\n", i);
	*ret = i;
	pthread_exit(ret);
}
```

```
void char_stat(char *argv)
{
        size_t length = 0;
        char *cptr = NULL;
        char *substr = NULL;
        char *brka = NULL;
        char *sep = "{}()[],;\" \n\t^";
        cptr = strdup(argv);
        for (substr = strtok_r(cptr, sep, &brka); substr; substr = strtok_r(NULL,
sep, &brka))
        {
                length = strlen(substr);
                for (int i = 0; i < length; i++)
                {
                        if (*substr < 256 && *substr > 1)
                        {
                                stat[*substr]++;
                        }
                substr++;
                }
        }
        free(cptr);
        return;
}

int main (int argc, char *argv[])
{
        pthread_t prod[100];
        pthread_t cons[100];
        int Nprod, Ncons;
        int rc;    long t;
        int *ret;
        int i;
        FILE *rfile;
        if (argc == 1) {
                printf("usage: ./prod_cons <readfile> #Producer #Consumer\n");
                exit (0);
        }
        buffer *buf = malloc(sizeof(buffer));
        memset(buf, 0, sizeof(buffer));
        so_t *share = malloc(sizeof(so_t));
```

```
memset(share, 0, sizeof(so_t));
// initialize stat
memset(stat, 0, sizeof(stat));
rfile = fopen((char *) argv[1], "r");
if (rfile == NULL) {
        perror("rfile");
        exit(0);
}
if (argv[2] != NULL) {
        Nprod = atoi(argv[2]);
        if (Nprod > 100) Nprod = 100;
        if (Nprod == 0) Nprod = 1;
} else Nprod = 1;
if (argv[3] != NULL) {
        Ncons = atoi(argv[3]);
        if (Ncons > 100) Ncons = 100;
        if (Ncons == 0) Ncons = 1;
} else Ncons = 1;

share->rfile = rfile;

for(i=0;i<Ncons;i++){
share->buf[i].line = NULL;
pthread_mutex_init(&share->buf[i].lock, NULL);
}
pthread_mutex_init(&STAT, NULL);
pthread_mutex_init(&con, NULL);
for (i = 0 ; i < Nprod ; i++)
        pthread_create(&prod[i], NULL, producer, share);
for (i = 0 ; i < Ncons ; i++)
        pthread_create(&cons[i], NULL, consumer, share);
printf("main continuing\n");

for (i = 0 ; i < Ncons ; i++) {
        rc = pthread_join(cons[i], (void **) &ret);
        printf("main: consumer_%d joined with %d\n", i, *ret);
}
for (i = 0 ; i < Nprod ; i++) {
        rc = pthread_join(prod[i], (void **) &ret);
        printf("main: producer_%d joined with %d\n", i, *ret);
}
```

```
        printf(" A : %8d\nB : %8d\nC : %8d\nD : %8d\nE : %8d\nF : %8d\nG :
%8d\nH : %8d\nI : %8d\nJ : %8d\nK : %8d\nL : %8d\nM : %8d\nN : %8d\nO :
%8d\nP : %8d\nQ : %8d\nR : %8d\nS : %8d\nT : %8d\nU : %8d\nV : %8d\nW :
%8d\nX : %8d\nY : %8d\nZ : %8d\n",
                        stat['A']+stat['a'],    stat['B']+stat['b'],    stat['C']+stat['c'],
stat['D']+stat['d'],  stat['E']+stat['e'],
                        stat['F']+stat['f'],    stat['G']+stat['g'],    stat['H']+stat['h'],
stat['I']+stat['i'],  stat['J']+stat['j'],
                        stat['K']+stat['k'],    stat['L']+stat['l'],    stat['M']+stat['m'],
stat['N']+stat['n'],  stat['O']+stat['o'],
                        stat['P']+stat['p'],    stat['Q']+stat['q'],    stat['R']+stat['r'],
stat['S']+stat['s'],  stat['T']+stat['t'],
                        stat['U']+stat['u'],    stat['V']+stat['v'],    stat['W']+stat['w'],
stat['X']+stat['x'],  stat['Y']+stat['y'],
                        stat['Z']+stat['z']);
        pthread_exit(NULL);
        exit(0);
}
```

<Multicons3.c> - Multi producer와 Multi consumer, buffer 간의 1:1:1 연결
time ./a.out FreeBSD9-orig.tar 2 2 실행할 경우 : 1m29.767s

```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <time.h>
#define MAX_STRING_LENGTH 30
#define ASCII_SIZE        256
int stat [ASCII_SIZE];
void char_stat(char* argv);
FILE *rfile;
int End = 0;

typedef struct sharedobject {
        int linenum;
        pthread_mutex_t lock;
        pthread_cond_t cond;
        char *line;
        char full;
        int in;
        int count;
}so_t;

void *producer(void *arg) {
        so_t *so = arg;
        int *ret = malloc(sizeof(int));
        int i = 0;
        int j = 0;
        char *line = NULL;
        size_t len = 0;
        ssize_t read = 0;

        while (1) {
                read = getdelim(&line, &len, '\n', rfile);
                pthread_mutex_lock(&so[so->in].lock);
                while(so->full == 1)
                {
                        pthread_cond_wait(&so->cond, &so->lock);
```

```
                }
                if (read == -1) {
                        so->full = 1;
                        so->line = NULL;
                        so->count = 1;
                        End = 1;
                        pthread_cond_signal(&so->cond);
                        pthread_mutex_unlock(&so->lock);
                        break;
                }
                so->linenum = i;
                so->line = strdup(line);        /* share the line */
                so->full = 1;
                i++;
                pthread_cond_signal(&so->cond);
                pthread_mutex_unlock(&so->lock);
        }
        free(line);
        line = NULL;
        printf("Prod_%x: %d lines\n", (unsigned int)pthread_self(), i);
        *ret = i;
        pthread_exit(ret);
}

void *consumer(void *arg) {
        so_t *so = arg;
        int *ret = malloc(sizeof(int));
        int i = 0;
        int len;
        char *line;
        while (1) {
                pthread_mutex_lock(&so->lock);
                while(so->full == 0)
                {
                        pthread_cond_wait(&so->cond, &so->lock);
                }
                line = so->line;
                if (so->line == NULL) {
                        so->full = 0;
                        pthread_cond_signal(&so->cond);
                        pthread_mutex_unlock(&so->lock);
```

```
                        break;
                }
                char_stat(line);
                len = strlen(line);
                //printf("Cons_%x: [%02d:%02d] %s",
                //      (unsigned int)pthread_self(), i, so->linenum, line);
                free(so->line);
                so->line = NULL;
                line = NULL;
                i++;
                so->full = 0;
                pthread_cond_signal(&so->cond);
                pthread_mutex_unlock(&so->lock);
        }
        printf("Cons: %d lines\n", i);
        *ret = i;
        pthread_exit(ret);
}

void char_stat(char *argv)
{
        size_t length = 0;
        char *cptr = NULL;
        char *substr = NULL;
        char *brka = NULL;
        char *sep = "{}()[],;\" \n\t^";
        cptr = strdup(argv);
        for (substr = strtok_r(cptr, sep, &brka); substr; substr = strtok_r(NULL,
sep, &brka))
        {
                length = strlen(substr);
                for (int i = 0; i < length; i++)
                {
                        if (*substr < 256 && *substr > 1)
                        {
                                stat[*substr]++;
                        }
                substr++;
                }
        }
        free(cptr);
```

```
        return;
}

int main (int argc, char *argv[])
{
        pthread_t prod[100];
        pthread_t cons[100];
        int Nprod, Ncons;
        int rc;    long t;
        int *ret;
        int i;
        if (argc == 1) {
                printf("usage: ./prod_cons <readfile> #Producer #Consumer\n");
                exit (0);
        }
        so_t *share[100];
        for(int i = 0; i<100; i++){
        share[i] = malloc(sizeof(so_t));
        memset(share[i], 0, sizeof(so_t));}
        // initialize stat
        memset(stat, 0, sizeof(stat));
        rfile = fopen((char *) argv[1], "r");
        if (rfile == NULL) {
                perror("rfile");
                exit(0);
        }
        if (argv[2] != NULL) {
                Nprod = atoi(argv[2]);
                if (Nprod > 100) Nprod = 100;
                if (Nprod == 0) Nprod = 1;
        } else Nprod = 1;
        if (argv[3] != NULL) {
                Ncons = atoi(argv[3]);
                if (Ncons > 100) Ncons = 100;
                if (Ncons == 0) Ncons = 1;
        } else Ncons = 1;

        for(i=0;i<Ncons;i++){
        share[i]->line = NULL;
        pthread_mutex_init(&share[i]->lock, NULL);
        pthread_cond_init(&share[i]->cond, NULL);
```

```
	}
	for (i = 0 ; i < Nprod ; i++)
		pthread_create(&prod[i], NULL, producer, share[i]);
	for (i = 0 ; i < Ncons ; i++)
		pthread_create(&cons[i], NULL, consumer, share[i]);
	printf("main continuing\n");

	for (i = 0 ; i < Ncons ; i++) {
		rc = pthread_join(cons[i], (void **) &ret);
		printf("main: consumer_%d joined with %d\n", i, *ret);
	}
	for (i = 0 ; i < Nprod ; i++) {
		rc = pthread_join(prod[i], (void **) &ret);
		printf("main: producer_%d joined with %d\n", i, *ret);
	}
	printf(" A : %8d\nB : %8d\nC : %8d\nD : %8d\nE : %8d\nF : %8d\nG :
%8d\nH : %8d\nI : %8d\nJ : %8d\nK : %8d\nL : %8d\nM : %8d\nN : %8d\nO :
%8d\nP : %8d\nQ : %8d\nR : %8d\nS : %8d\nT : %8d\nU : %8d\nV : %8d\nW :
%8d\nX : %8d\nY : %8d\nZ : %8d\n",
			stat['A']+stat['a'],	stat['B']+stat['b'],		stat['C']+stat['c'],
stat['D']+stat['d'],	stat['E']+stat['e'],
			stat['F']+stat['f'],	stat['G']+stat['g'],		stat['H']+stat['h'],
stat['I']+stat['i'],	stat['J']+stat['j'],
			stat['K']+stat['k'],	stat['L']+stat['l'],		stat['M']+stat['m'],
stat['N']+stat['n'],	stat['O']+stat['o'],
			stat['P']+stat['p'],	stat['Q']+stat['q'],		stat['R']+stat['r'],
stat['S']+stat['s'],	stat['T']+stat['t'],
			stat['U']+stat['u'],	stat['V']+stat['v'],		stat['W']+stat['w'],
stat['X']+stat['x'],	stat['Y']+stat['y'],
			stat['Z']+stat['z']);

	pthread_exit(NULL);
	exit(0);
}
```