Operating System Project #2 Report

Synchronization Problem

12191656 이채연

I. 개발환경

Synchronization Problem을 해결하는 프로그램을 구현하기 위해서 Window 내에서 가상머신을 이용하여 Linux를 설치하였습니다. VirtualBox 라는 가상머신을 이용하였고, VirtualBox 6.1.22 platform packages 버전을 사용하였습니다. 그 다음, Ubuntu 20.04.2.0 버전의 리눅스 설치 파일을 받은 뒤, 가상머신을 실행하여 "Linux_Ubuntu" 라는 이름의 새로운 가상머신을 실행하여 리눅스를 설치하였습니다.
이렇게 실행한 Linux OS에서의 CPU 정보, memory 정보, 그리고 Linux 배포판 정보 및 커널 정보는 다음과 같습니다.

```
chaeyeon@chaeyeon-VirtualBox:~$ cat /proc/cpuinfo
processor       : 0
vendor_id       : GenuineIntel
cpu family      : 6
model           : 61
model name      : Intel(R) Core(TM) i5-5200U CPU @ 2.20GHz
stepping        : 4
cpu MHz         : 2194.918
cache size      : 3072 KB
physical id     : 0
siblings        : 1
core id         : 0
cpu cores       : 1
apicid          : 0
initial apicid  : 0
fpu             : yes
fpu_exception   : yes
cpuid level     : 20
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov
 pat pse36 clflush mmx fxsr sse sse2 ht syscall nx rdtscp lm constant_tsc rep_g
ood nopl xtopology nonstop_tsc cpuid tsc_known_freq pni pclmulqdq monitor ssse3
 cx16 pcid sse4_1 sse4_2 x2apic movbe popcnt aes xsave avx rdrand hypervisor la
hf_lm abm 3dnowprefetch invpcid_single pti fsgsbase avx2 invpcid rdseed md_clea
r flush_l1d
bugs            : cpu_meltdown spectre_v1 spectre_v2 spec_store_bypass l1tf mds
 swapgs itlb_multihit srbds
bogomips        : 4389.83
clflush size    : 64
cache_alignment : 64
address sizes   : 39 bits physical, 48 bits virtual
power management:
```

```
chaeyeon@chaeyeon-VirtualBox:~$ cat /proc/meminfo
MemTotal:        1004312 kB
MemFree:           72280 kB
MemAvailable:     252880 kB
Buffers:           14952 kB
Cached:           286568 kB
SwapCached:        12516 kB
Active:           501628 kB
Inactive:         289896 kB
Active(anon):     294400 kB
Inactive(anon):   204804 kB
Active(file):     207228 kB
Inactive(file):    85092 kB
Unevictable:           0 kB
Mlocked:               0 kB
SwapTotal:        459260 kB
SwapFree:         383444 kB
Dirty:                 0 kB
Writeback:             0 kB
AnonPages:        483936 kB
Mapped:           163844 kB
Shmem:              9200 kB
KReclaimable:      31584 kB
Slab:              88276 kB
SReclaimable:      31584 kB
SUnreclaim:        56692 kB
KernelStack:        6268 kB
PageTables:        11232 kB
NFS_Unstable:          0 kB
Bounce:                0 kB
WritebackTmp:          0 kB
CommitLimit:      961416 kB
Committed_AS:    3354328 kB
VmallocTotal:   34359738367 kB
VmallocUsed:       30524 kB
VmallocChunk:          0 kB
Percpu:              644 kB
HardwareCorrupted:     0 kB
AnonHugePages:         0 kB
ShmemHugePages:        0 kB
ShmemPmdMapped:        0 kB
FileHugePages:         0 kB
FilePmdMapped:         0 kB
HugePages_Total:       0
HugePages_Free:        0
HugePages_Rsvd:        0
HugePages_Surp:        0
Hugepagesize:       2048 kB
Hugetlb:               0 kB
DirectMap4k:      139200 kB
DirectMap2M:      909312 kB
```

```
chaeyeon@chaeyeon-VirtualBox:~$ cat /proc/version
Linux version 5.8.0-50-generic (buildd@lgw01-amd64-030) (gcc (Ubuntu 9.3.0-17ub
untu1~20.04) 9.3.0, GNU ld (GNU Binutils for Ubuntu) 2.34) #56~20.04.1-Ubuntu S
MP Mon Apr 12 21:46:35 UTC 2021
```

II. Synchronization Problem을 해결하는 프로그램 설계 및 구현 내용

이번 프로젝트는 동기화 문제를 보고 x의 값이 버퍼의 크기로서 최대 크기가 30일 때 생산자 소비자 문제로 해결하는 프로그램을 구현하는 것입니다.

먼저 리눅스에서 synch_pthread.c 와 synch_semaphore.c 라고 하는 c 소스파일을 만들었습니다.
1)문제를 해결하는 코드는 synch_pthread.c 소스파일에 저장되어 있으며,
2)문제를 해결하는 코드는 synch_semaphore.c 소스파일에 저장되어 있습니다.

```
chaeyeon@chaeyeon-VirtualBox:~$ vi synch_pthread.c
chaeyeon@chaeyeon-VirtualBox:~$ vi synch_semaphore.c
chaeyeon@chaeyeon-VirtualBox:~$ ls
Desktop    Makefile   myshell.h    Pictures         synch_semaphore.c
Documents  Music      myshell.o    Public           Templates
Downloads  myshell.c  myshell.out  synch_pthread.c  Videos
```

그리고 make 라고 하는 파일 관리 유틸리티를 사용하기 위해서 Makefile을 만들었습니다.

```
chaeyeon@chaeyeon-VirtualBox:~$ vi Makefile
```

Makefile 코드는 다음과 같습니다.

```
CC = gcc
CFLAGS = -lpthread

OBJS1 = synch_pthread.o
OBJS2 = synch_semaphore.o
SRCS = $(OBJS1: .o=.c) $(OBJS2: .o=.c)

TARGET: synch_pthread synch_semaphore

all: $(TARGET)

synch_pthread : $(OBJS1)
                $(CC) -o synch_pthread $(OBJS1) $(CFLAGS)
synch_semaphore : $(OBJS2)
                $(CC) -o synch_semaphore $(OBJS2) $(CFLAGS)

dep :
                gccmakedep $(SRCS)

clean :
        -rm -f $(OBJS1) $(OBJS2) $(TARGET)

rebuild: clean all
```

Target 파일을 각 두 1) 2)문제에 대한 실행 파일인 synch_pthread와 synch_semaphore로 정하고 이 바이너리 파일을 만들기 위한 목적파일인 synch_pthread.o와 synch_semaphore.o 를 만들어주었습니다.
all option을 통해 최종으로 만들고자 하는 것이 실행파일인 synch_pthread와 synch_semaphore라는 것을 표시합니다.

1)pthread_mutex_lock와 pthread_cond_wait/pthread_cond_signal을 이용하여 수정하시오.

이 문제를 해결하기 위한 synch_pthread.c 코드는 다음과 같습니다.

```c
#include <pthread.h>
#include <stdio.h>
#define ITER 100000
#define MAX 30

void *producer(void *arg);
void *consumer(void *arg);
void put();
int get();
int buffer[MAX];
int fill_cnt = 0;
int use_cnt = 0;
int x = 0;
pthread_cond_t empty;
pthread_cond_t fill;
pthread_mutex_t m;

int main(){
        pthread_t tid1, tid2;
        pthread_cond_init(&empty, NULL);
        pthread_cond_init(&fill, NULL);
        pthread_mutex_init(&m, NULL);
        pthread_create(&tid1, NULL, producer, NULL);
        pthread_create(&tid2, NULL, consumer, NULL);

        pthread_join(tid1, NULL);
        pthread_join(tid2, NULL);
```

```
        if(x!=0)
                printf("BOOM! counter=%d\n", x);
        else
                printf("OK counter=%d\n", x);

        pthread_cond_destroy(&empty);
        pthread_cond_destroy(&fill);
        pthread_mutex_destroy(&m);
}

/*thread routine*/
void * producer(void *arg){
int i, val;
for(i=0; i<ITER; i++){
        pthread_mutex_lock(&m);
        while (x==30)
                pthread_cond_wait(&empty, &m);
        val = x;
        printf("%u: %d\n", (unsigned int)pthread_self(), val);
        put(val);
        x = val+1;
        pthread_cond_signal(&fill);
        pthread_mutex_unlock(&m);
}
return NULL;
}
void *consumer(void *arg){
int i, val;
for(i=0; i<ITER; i++){
        pthread_mutex_lock(&m);
        while(x == 0)
                pthread_cond_wait(&fill, &m);
        val = x;
        printf("%u: %d\n", (unsigned int)pthread_self(), val);
        get();
        x = val-1;
        pthread_cond_signal(&empty);
        pthread_mutex_unlock(&m);
}
return NULL;
}

void put(int item){
        buffer[fill_cnt] = item;
        fill_cnt = (fill_cnt+1)%MAX;
        return;
}

int get(){
        int tmp = buffer[use_cnt];
        use_cnt = (use_cnt + 1)%MAX;
        return tmp;
}

-- INSERT --                                            83,1          Bot
```

producer 함수와 consumer 함수를 보면 shared data인 x와 buffer에 접근하는 critical section을 상호배제 시키기 위해 critical section을 P(m) operation과 V(m)operation으로 감쌌습니다. 그리고 x를 버퍼에 들어있는 value(item)의 수라고 생각하였습니다.

producer는 x=30일 때 즉, buffer가 꽉 차있을때 consumer에 의해 buffer가 비도록 기다리는 코드를 작성하였습니다. while문 돌도록 하여 x가 30인지, 버퍼에 빈 slot이 있는지를 rechecking 할 수 있도록 하였습니다. critical section이 끝난 뒤, producer는 item을 buffer에 추가했으므로 consumer함수에서 buffer가 차기를 기다리는 코드로 가서 깨어줍니다(pthread_cond_wait(&fill, &m)).

consumer는 x=0일 때 즉, buffer가 비어있을 때 producer에 의해 buffer가 채워지도록 기다리는 코드를 작성하였습니다. 마찬가지로 while문을 돌도록 하여 x가 0인지, 버퍼에 item이 채워져 있는지를 rechecking 할 수 있도록 하였습니다. critical section이 끝난 뒤, consumer는 item을 buffer에서 빼왔으므로 producer함수에서 buffer가 비기를 기다리는 코드로 가서 깨워줍니다 (pthread_cond_wait(&empty, &m)).

2)세마포어를 사용하여 위의 동기화 문제를 다음과 같이 수정하라.

이 문제를 해결하기 위한 synch_semaphore.c 코드는 다음과 같습니다.

```c
#include <pthread.h>
#include <semaphore.h>
#include <stdio.h>
#define ITER 100000
#define MAX 30

void *producer(void *arg);
void *consumer(void *arg);
void put();
int get();
int buffer[MAX];
int fill_cnt = 0;
int use_cnt = 0;
int x = 0;
sem_t m, fill, empty;

int main(){
        pthread_t tid1, tid2;
        sem_init(&m, 0, 1);
        sem_init(&fill, 0, 0);
        sem_init(&empty, 0, MAX);
        pthread_create(&tid1, NULL, producer, NULL);
        pthread_create(&tid2, NULL, consumer, NULL);
        pthread_join(tid1, NULL);
        pthread_join(tid2, NULL);

        if(x!=0)
                printf("BOOM! counter=%d\n",x);
        else
                printf("OK counter=%d\n", x);
        sem_destroy(&m);
}

void *producer(void *arg){
int i, val;
for(i = 0; i<ITER; i++){
        sem_wait(&empty);
        sem_wait(&m);
        val = x;
        printf("%u: %d\n", (unsigned int)pthread_self(), val);
        put(val);
        x = val+1;
        sem_post(&m);
        sem_post(&fill);
}
return NULL;
}

void *consumer(void *arg){
int i, val;
for(i=0; i<ITER; i++){
        sem_wait(&fill);
        sem_wait(&m);
        val = x;
        printf("%u: %d\n", (unsigned int)pthread_self(), val);
        get();
        x = val-1;
        sem_post(&m);
        sem_post(&empty);
}
return NULL;
}

void put(int item){
        buffer[fill_cnt] = item;
        fill_cnt = (fill_cnt + 1)%MAX;
        return;
}

int get(){
        int tmp = buffer[use_cnt];
        use_cnt = (use_cnt + 1)%MAX;
        return tmp;
}
-- INSERT --                                                69,1          Bot
```

세마포어 변수인 m, fill, empty를 선언하였고 m은 상호배제를 위한 것이므로 1로 초기화 하였으며, fill이라는 세마포어 변수는 consumer 입장에서 buffer에 데이터가 들어오기를 기다리는 상황에서 쓰이므로 buffer에 들어있는 item의 수를 의미합니다. 그래서 0으로 초기화 하였고, empty라는 세마포어 변수는 producer 입장에서 buffer에 빈 slot이 있기를 기다리는 상황에서 쓰이므로 buffer에 item이 들어갈 수 있는 자리 수를 의미합니다. 그래서 버퍼의 최대 사이즈인 30으로 초기화 하였습니다.

또한, 1)문제와 마찬가지로 x를 버퍼에 들어있는 value(item)의 수라고 생각하고, producer 함수와 consumer 함수를 보면 shared data인 x와 buffer에 접근하는 critical section을 상호배제 시키기 위해 critical section을 P(m) operation과 V(m)operation으로 감쌌습니다.
여기서 condition variable 코드 전에 상호배제 코드가 들어가면 세마포어는 lock을 걸고 자러가는 상황이 발생할 수 있으므로 상호배제 코드 전에 condition variable 코드를 넣어 그런 상황이 발생하는 것을 방지하였습니다.
Condition variable 코드 작동과정은 1)문제와 동일합니다.

III. 실행 결과

Makefile 실행 방법입니다.

```
chaeyeon@chaeyeon-VirtualBox:~$ make
gcc -lpthread    -c -o synch_pthread.o synch_pthread.c
gcc -o synch_pthread synch_pthread.o -lpthread
gcc -lpthread    -c -o synch_semaphore.o synch_semaphore.c
gcc -o synch_semaphore synch_semaphore.o -lpthread
```

1)pthread_mutex_lock와 pthread_cond_wait/pthread_cond_signal을 이용하여 동기화 문제를 생산자 소비자 문제로 해결한 결과는 다음과 같습니다.

```
chaeyeon@chaeyeon-VirtualBox:~$ ./synch_pthread
3120097024: 15              3111704320: 1
3120097024: 16              3120097024: 0
3120097024: 17              3111704320: 1
3120097024: 18              3120097024: 0
3111704320: 19              3111704320: 1
3111704320: 18              3120097024: 0
3111704320: 17              3111704320: 1
3111704320: 16              3120097024: 0
3111704320: 15              3111704320: 1
3111704320: 14              3120097024: 0
3111704320: 13              3111704320: 1
3111704320: 12              3120097024: 0
3111704320: 11              3111704320: 1
3111704320: 10              3120097024: 0
3111704320: 9               3111704320: 1
3111704320: 8               3120097024: 0
3111704320: 7               3111704320: 1
3111704320: 6               3120097024: 0
3111704320: 5               3111704320: 1
3111704320: 4               3120097024: 0
3111704320: 3               3111704320: 1
3111704320: 2               3120097024: 0
3111704320: 1               3111704320: 1
3120097024: 0               3120097024: 0
3120097024: 1               3111704320: 1
3120097024: 2               3120097024: 0
3120097024: 3               3111704320: 1
3120097024: 4               OK counter=0
3120097024: 5               chaeyeon@chaeyeon-VirtualBox:~$
```

2)세마포어를 사용하여 동기화 문제를 생산자 소비자 문제로 해결한 결과는 다음과 같습니다.

```
chaeyeon@chaeyeon-VirtualBox:~$ ./synch_semaphore
614401792: 27               606009088: 21
614401792: 28               606009088: 20
614401792: 29               606009088: 19
606009088: 30               606009088: 18
606009088: 29               606009088: 17
606009088: 28               606009088: 16
606009088: 27               606009088: 15
606009088: 26               606009088: 14
606009088: 25               606009088: 13
606009088: 24               606009088: 12
606009088: 23               606009088: 11
606009088: 22               606009088: 10
606009088: 21               606009088: 9
606009088: 20               606009088: 8
606009088: 19               606009088: 7
606009088: 18               606009088: 6
606009088: 17               606009088: 5
606009088: 16               606009088: 4
606009088: 15               606009088: 3
606009088: 14               606009088: 2
606009088: 13               606009088: 1
606009088: 12               614401792: 0
606009088: 11               614401792: 1
606009088: 10               614401792: 2
606009088: 9                606009088: 3
606009088: 8                606009088: 2
606009088: 7                606009088: 1
606009088: 6                OK counter=0
606009088: 5                chaeyeon@chaeyeon-VirtualBox:~$
```

1)과 2)의 실행결과를 봤을 때 동기화 문제가 생산자 소비자 문제로 잘 해결된 것을 확인할 수 있습니다.