Lab 6: Process Synchronization and Mutex (I)

1.Objective

- Study the process synchronization and mutex

2.Syllabus

- Understanding the concepts of process synchronization and mutex
- Implementing the process synchronization and mutex using C or C++

3.Prerequisite

- C or C++ language
- Computer which has ran Linux system (e.g. Ubuntu)
- Understand the concepts of process communications

4.Concepts and Principles of Process communication

Please refer to the chapter 5 of the text book. The main contents have been lectured in the fourth week, 11/14/2016. The slide of this chapter can be found in the course website: http://www.thinkmesh.net/ose/.

5.Experimental Contents

5.1 Producer-Consumer Problem (Semaphore)

The following program adopts the semaphore to solve the producer-consumer problem. You should save the codes into different files: "**producer.c**" and "**consumer.c**". Please do not **copy-and-paste** the codes.

After you can run the following programs, you **MUST** modify them and consider multiple producers and consumers. If the modified programs can run, you should present your result to me. If not, tell me why.

```
//producer.c
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/sem.h>
#define SHMKEY 70
#define SEMKEY 80
#define K          1024
typedef int arr[16][256];
int shmid;
int semid;
void cleanup();
int main()
{
      int i, in=0;
      char *addr;
```

```
        arr    *arrp;
        struct sembuf ops1={0,-1,SEM_UNDO},    ops2={1, 1,SEM_UNDO};
        for (i=0; i<31; i++)
                signal(i,cleanup);
        shmid = shmget(SHMKEY, 16*K, 0777|IPC_CREAT);
        addr = shmat(shmid, 0, 0);
        arrp = (arr *)addr;
        semid = semget(SEMKEY, 2, 0777|IPC_CREAT);
        semctl(semid,0,SETVAL,16);
        semctl(semid,1,SETVAL,0);
        for (i=0; i<256; i++)
        {
                semop(semid, &ops1, 1);
                (*arrp)[in][0] = i;
                in = (in+1) % 16;
                semop(semid, &ops2, 1);
        }
        sleep(1);
        return 0;
}

void cleanup()
{
        shmctl(shmid,IPC_RMID,0);
        semctl(semid,0,IPC_RMID,0);
        exit(0);
}
```

```
//consumer.c
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/sem.h>
#define SHMKEY 70
#define SEMKEY 80
#define K           1024
typedef int arr[16][256];
int shmid;
int semid;
int main()
{
        int i, in=0;
        char *addr;
        arr    *arrp;
        struct sembuf ops1={1,-1,SEM_UNDO}, ops2={0, 1,SEM_UNDO};
```

```
    shmid = shmget(SHMKEY, 16*K, 0777);
    addr = shmat(shmid, 0, 0);
    arrp = (arr *)addr;
    semid = semget(SEMKEY, 2, 0777);
    for (i=0; i<256; i++)
    {
        semop(semid, &ops1, 1);
        printf("sem2: %d = %d\n",i,(*arrp)[in][0]);
        in = (in+1) % 16;
        semop(semid, &ops2, 1);
    }
    return 0;
}
```

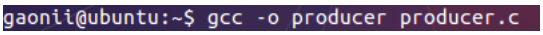
How to Run: gcc -o producer producer.c
gcc -o consumer consumer.c
./producer&
./consumer

The detailed procedure:

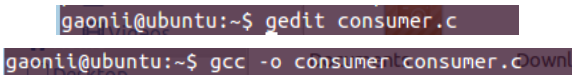
(1) Open the terminal, input a command "**gedit producer.c**" to create a new document named **producer.c** and open it.

(2) Input above code "**producer.c**" and save it.

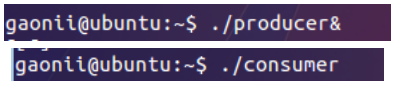
(3) Use the command "**gcc -o consumer consumer.c**" to generate an executable file named "**producer**".



(4) Repeat the step (1) (2) (3). Input above code "**consumer.c**" and generate an executable file named "**consumer**".



(5) Input commands in order:
./producer&
./consumer



Then you can get the result.

5.2 Readers-Writers' Problem (Synchronized Lock)

Using the share lock to solve the readers-writers' problem. The file name is "**readers_writers.c**". **Note:** please modify the program, and consider multiple readers and writers. If it can run, show me your result. If not, tell me why.

```
#include <pthread.h>
#include <stdio.h>
#include <unistd.h>
```

```c
#define _READR_NUM_ 3
#define _WRITER_NUM_ 2
pthread_rwlock_t lock;
int buf = 0;

void *read(void* _val)
{
	pthread_detach(pthread_self());
	while(1)
	{
		if(pthread_rwlock_tryrdlock(&lock) != 0)
		{
			printf(" writer is writting .. readr is waitting\n");
		}
		else
		{
			printf("readr is :%u, read val id: %d\n",
					pthread_self(), buf);
			pthread_rwlock_unlock(&lock);
		}
		sleep(1);
	}
}

void *write(void* _val)
{
	pthread_detach(pthread_self());
	while(1)
	{
		if(pthread_rwlock_tryrdlock(&lock) != 0)
		{
			printf(" readr is reading .. writer is waiting\n");
		}
		else
		{
		buf = rand();
			printf("writer is :%u, write val id: %d\n",
					pthread_self(), buf);
			pthread_rwlock_unlock(&lock);
		}
		sleep(1);
	}
}
```

```
int main()
{
	pthread_rwlock_init(&lock, NULL);
	pthread_t id;
	int i = 0;
	for(i = 0; i < _WRITER_NUM_; i++)
		pthread_create(&id, NULL, write, NULL);
	for(i = 0; i < _READR_NUM_; i++)
		pthread_create(&id, NULL, read, NULL);
	sleep(10);
	return 0;
}
```

How to Run: gcc -o reader_writer reader_writer.c -lpthread
./reader_writer

The detailed procedures:

(1) Open the terminal, input a command "**gedit reader_writer.c**" to create a new document named **reader_writer.c** and open it.

```
gaonii@ubuntu:~$ gedit reader_writer.c
```

(2) Input above codes "**reader_writer.c**" and save it.

(3) Use the command "**gcc -o reader_writer reader_writer.c -lpthread**" to generate an executable file named "**reader_writer**".

```
gaonii@ubuntu:~$ gcc -o reader_writer reader_writer.c -lpthread
```

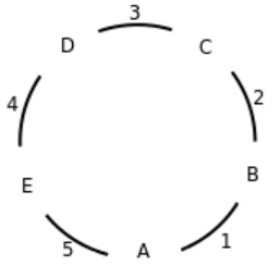
(4) Run the command "**./reader_writer**"

```
gaonii@ubuntu:~$ ./readr_writer
```

Then you can get the result.

5.3 Dinning-Philosophers' Problem (Synchronized Lock)

Suppose five philosophers' number is A, B, C, D, E and have five knives' number is 1, 2, 3, 4, 5. Shown as follows:



Please solve the dinning-philosophers' problem. The file name is "**dinner.c**". **Note:** please modify the program, and consider cause deadlock. If it can run, show me your

result. If not, tell me why.

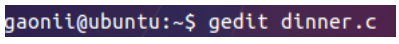
```
#include <stdio.h>
#include <stdlib.h>
#include <memory.h>
#include <pthread.h>
#include <errno.h>
#include <math.h>

pthread_mutex_t chopstick[6] ;
void *eat_think(void *arg)
{
	char phi = *(char *)arg;
	int left,right;
	switch (phi){
		case 'A':
			left = 5;
			right = 1;
			break;
		case 'B':
			left = 1;
			right = 2;
			break;
		case 'C':
			left = 2;
			right = 3;
			break;
		case 'D':
			left = 3;
			right = 4;
			break;
		case 'E':
			left = 4;
			right = 5;
			break;
	}

	int i;
	for(;;){
		usleep(3);
		pthread_mutex_lock(&chopstick[left]);
		printf("Philosopher %c fetches chopstick %d\n", phi, left);
		if (pthread_mutex_trylock(&chopstick[right]) == EBUSY){
			pthread_mutex_unlock(&chopstick[left]);
```

```
                continue;
            }

        //    pthread_mutex_lock(&chopstick[right]);
            printf("Philosopher %c fetches chopstick %d\n", phi, right);
            printf("Philosopher %c is eating.\n",phi);
            usleep(3);
            pthread_mutex_unlock(&chopstick[left]);
            printf("Philosopher %c release chopstick %d\n", phi, left);
            pthread_mutex_unlock(&chopstick[right]);
            printf("Philosopher %c release chopstick %d\n", phi, right);

        }
}
int main(){
        pthread_t A,B,C,D,E;

        int i;
        for (i = 0; i < 5; i++)
                pthread_mutex_init(&chopstick[i],NULL);
        pthread_create(&A,NULL, eat_think, "A");
        pthread_create(&B,NULL, eat_think, "B");
        pthread_create(&C,NULL, eat_think, "C");
        pthread_create(&D,NULL, eat_think, "D");
        pthread_create(&E,NULL, eat_think, "E");

        pthread_join(A,NULL);
        pthread_join(B,NULL);
        pthread_join(C,NULL);
        pthread_join(D,NULL);
        pthread_join(E,NULL);
        return 0;
}
```

How to Run: gcc -o dinner dinner.c -lpthread
./dinner

The detailed procedures:

(1) Open the terminal, input the command "**gedit dinner.c**" to create a new file named **dinner.c** and open it.



(2) Input above code "**dinner.c**" and save it.

(3) Using the command "**gcc -o dinner dinner.c -lpthread**" to generate an executable file named "**dinner**".

```
gaonii@ubuntu:~$ gcc -o dinner dinner.c  -lpthread
```

(4) Run the compiled program using "**./dinner**".

```
gaonii@ubuntu:~$ ./dinner
```

Then you can get the result.

5.4 Readers-Writers' Problem (Semaphore)

Using the semaphore to solve the readers-writers' problem. The file name is "**readers_writers2.c**". **Note:** please modify the program, and consider semaphore. If it can run, show me your result. If not, tell me why.

```
#include <stdio.h>
#include <stdlib.h>
#include <memory.h>
#include <pthread.h>
#include <errno.h>
#include <math.h>
#include <semaphore.h>

sem_t rmutex;
sem_t wmutex;
int wnum=0;
int rnum=0;
int buf[3]={0};

void read()
{
   rnum++;
   printf("reader numer %d\n", (rnum%3)+1);
   sleep(2);
}

void write()
{
      wnum++;
      printf("write number %d\n",(wnum%3)+1);
      sleep(2);
}

void *reader(void *arg)
{
      while(1)
      {
            sem_wait(&rmutex);
            if(rnum==0)
            {
```

```
                sem_wait(&wmutex);
        }
        rnum++;
        sem_post(&rmutex);

        read();

        sem_wait(&rmutex);
        rnum--;
        if(rnum==0)
        {
                sem_post(&wmutex);
        }
        sem_post(&rmutex);
    }
    pthread_exit(NULL);
}

void *writer(void *arg)
{
    while(1)
    {
        sem_wait(&wmutex);
        write();
        sem_post(&wmutex);
    }
    pthread_exit(NULL);
}

int main()
{
    sem_init(&rmutex,0,1);
    sem_init(&wmutex,0,1);

    pthread_t tid;
    int ret=0,i;
    for(i=0;i<3;i++)
    {
        pthread_create(&tid,NULL,writer,NULL);
        if(ret<0)
        {
                perror("pthread_create");
                exit(0);
        }
```

```
    }
    for(i=0;i<5;i++)
    {
        pthread_create(&tid,NULL,reader,NULL);
        if(ret<0)
        {
            perror("pthread_create");
            exit(0);
        }
    }

    pthread_join(tid,NULL);

    return 0;
}
```

How to Run: gcc -o reader_writer2 reader_writer2.c -lpthread
./reader_writer2

5.5 Dinning-Philosophers' Problem (Semaphore)

Please program to implement dinning-philosophers' problem. The file name is "**dinner2.c**". **Note:** please modify the program, and consider semaphore. If it can run, show me your result. If not, tell me why.

```
#include <stdio.h>
#include <stdlib.h>
#include <memory.h>
#include <pthread.h>
#include <errno.h>
#include <math.h>
#include <semaphore.h>

sem_t room;
sem_t chip[5];

void *philosopher(void *arg)
{
    int i=0;
    i=*((int *)arg);
    while(1)
    {
        printf("philosoper %d thinking\n",i);
        sem_wait(&room);
```

```
            sem_wait(&chip[i]);
            sem_wait(&chip[(i+1)%5]);
            printf("philosopher %d eating\n",i);
            sem_post(&chip[(i+1)%5]);
            sem_post(&chip[i]);
            sem_post(&room);
            sleep(2);
        }
        pthread_exit(NULL);
}

int main()
{
        sem_init(&room,0,4);
        int i=0;
        int arg[5]={0};
        for(i=0;i<5;i++)
        {
            sem_init(chip+i,0,1);
            arg[i]=i;
        }

        int ret=0;
        pthread_t tid;
        for(i=0;i<5;i++)
        {
            ret=pthread_create(&tid,NULL,philosopher,(void *)&arg[i]);
            if(ret<0)
            {
                perror("pthread_create");
                exit(0);
            }
        }

#if 0
        while(1)
        {
            sleep(2);
        }
#endif

        pthread_join(tid,NULL);

        return 0;
```

}

How to Run: gcc -o dinner2 dinner2.c -lpthread
./dinner2

6.Conclusion

In this chapter, six experiments have been complete. They have shown how to synch and mutex with processes.