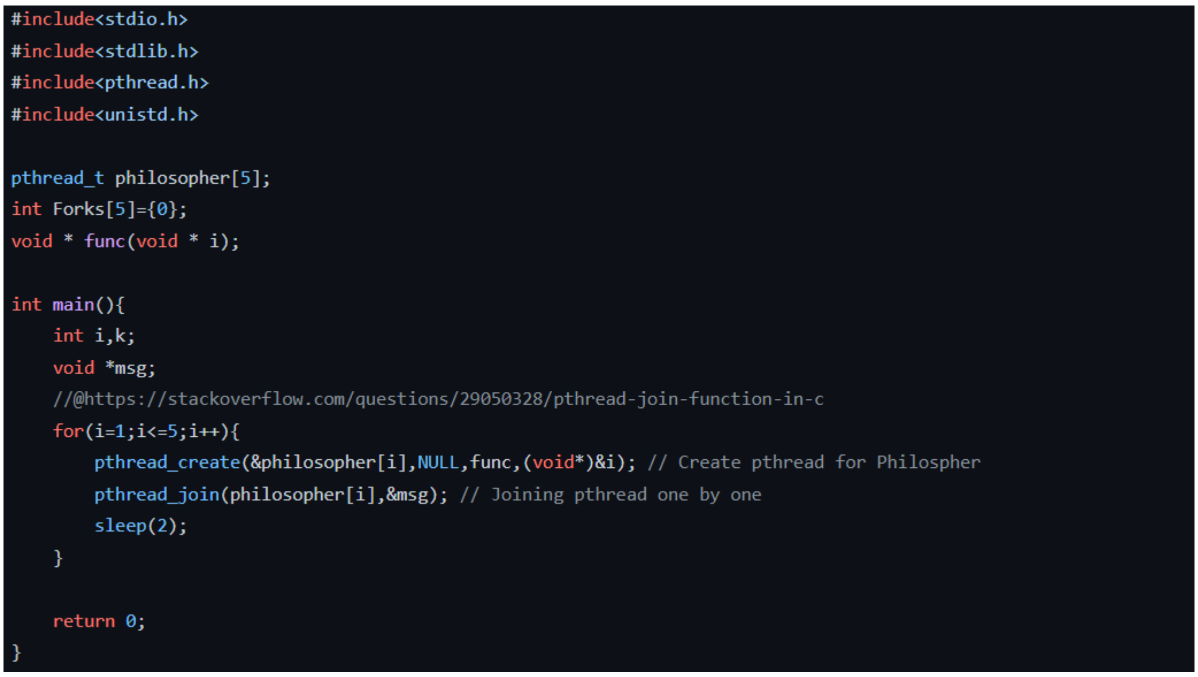
PART-1

A1:

No synchronization

Int main():



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

pthread_t philosopher[5];
int Forks[5]={0};
void * func(void * i);

int main(){
    int i,k;
    void *msg;
    //@https://stackoverflow.com/questions/29050328/pthread-join-function-in-c
    for(i=1;i<=5;i++){
        pthread_create(&philosopher[i],NULL,func,(void*)&i); // Create pthread for Philospher
        pthread_join(philosopher[i],&msg); // Joining pthread one by one
        sleep(2);
    }

    return 0;
}
```

Forks[i] array in gloabal used for available forks

A2:

For synchronization

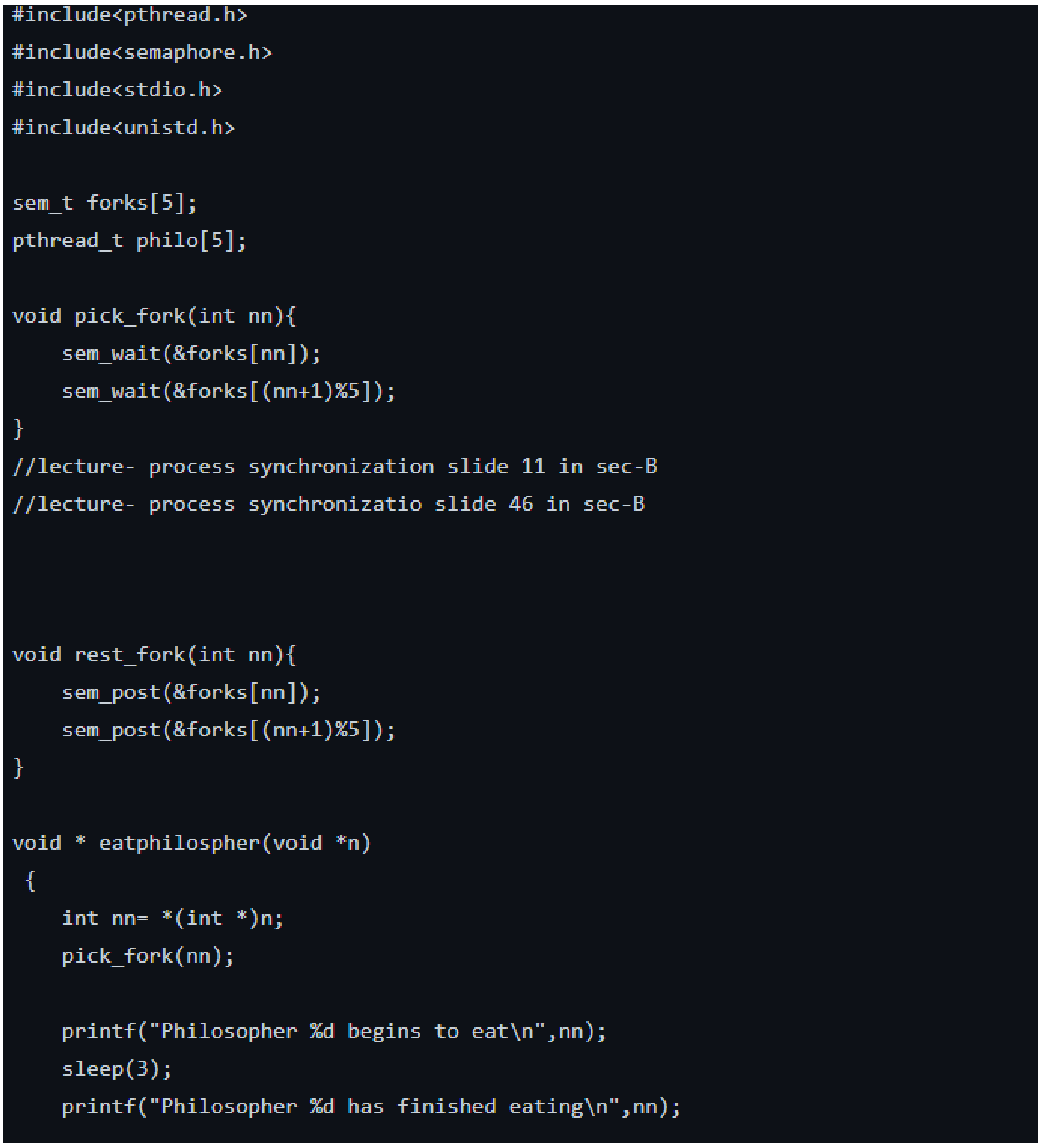
preventing deadlock Semaphore is used

Philosphers are picking forks first from left and right

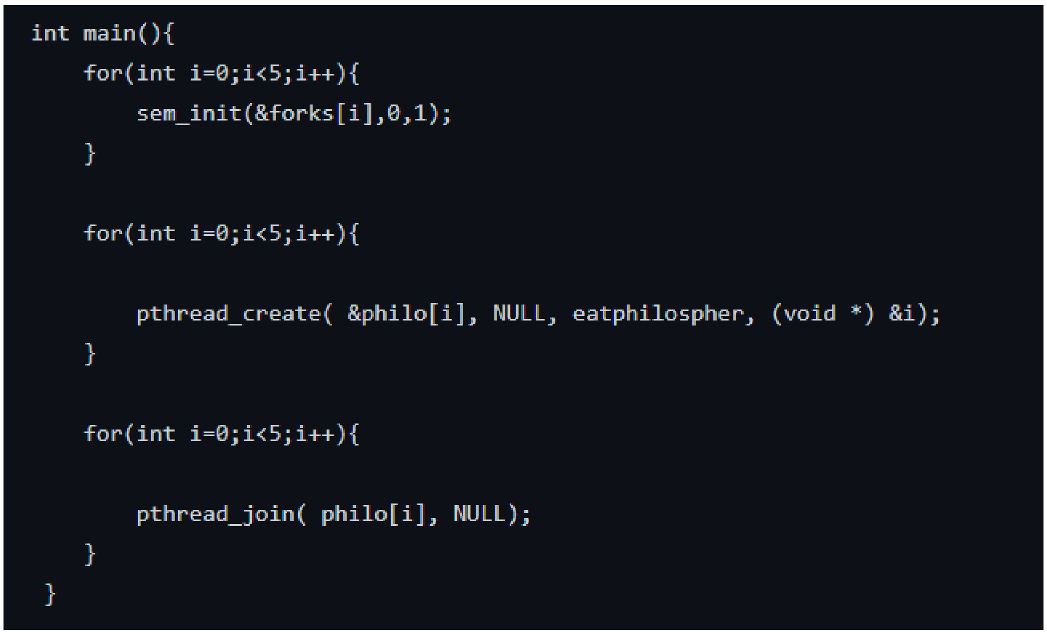
Forks are in global resources

sem_open() ,sem_unlink(), sem post used

No synchronization used



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

sem_t forks[5];
pthread_t philo[5];

void pick_fork(int nn){
    sem_wait(&forks[nn]);
    sem_wait(&forks[(nn+1)%5]);
}
//lecture- process synchronization slide 11 in sec-B
//lecture- process synchronizatio slide 46 in sec-B



void rest_fork(int nn){
    sem_post(&forks[nn]);
    sem_post(&forks[(nn+1)%5]);
}

void * eatphilospher(void *n)
 {
    int nn= *(int *)n;
    pick_fork(nn);

    printf("Philosopher %d begins to eat\n",nn);
    sleep(3);
    printf("Philosopher %d has finished eating\n",nn);
```



```c
 int main(){
     for(int i=0;i<5;i++){
         sem_init(&forks[i],0,1);
     }

     for(int i=0;i<5;i++){

         pthread_create( &philo[i], NULL, eatphilospher, (void *) &i);
     }

     for(int i=0;i<5;i++){

         pthread_join( philo[i], NULL);
     }
  }
```

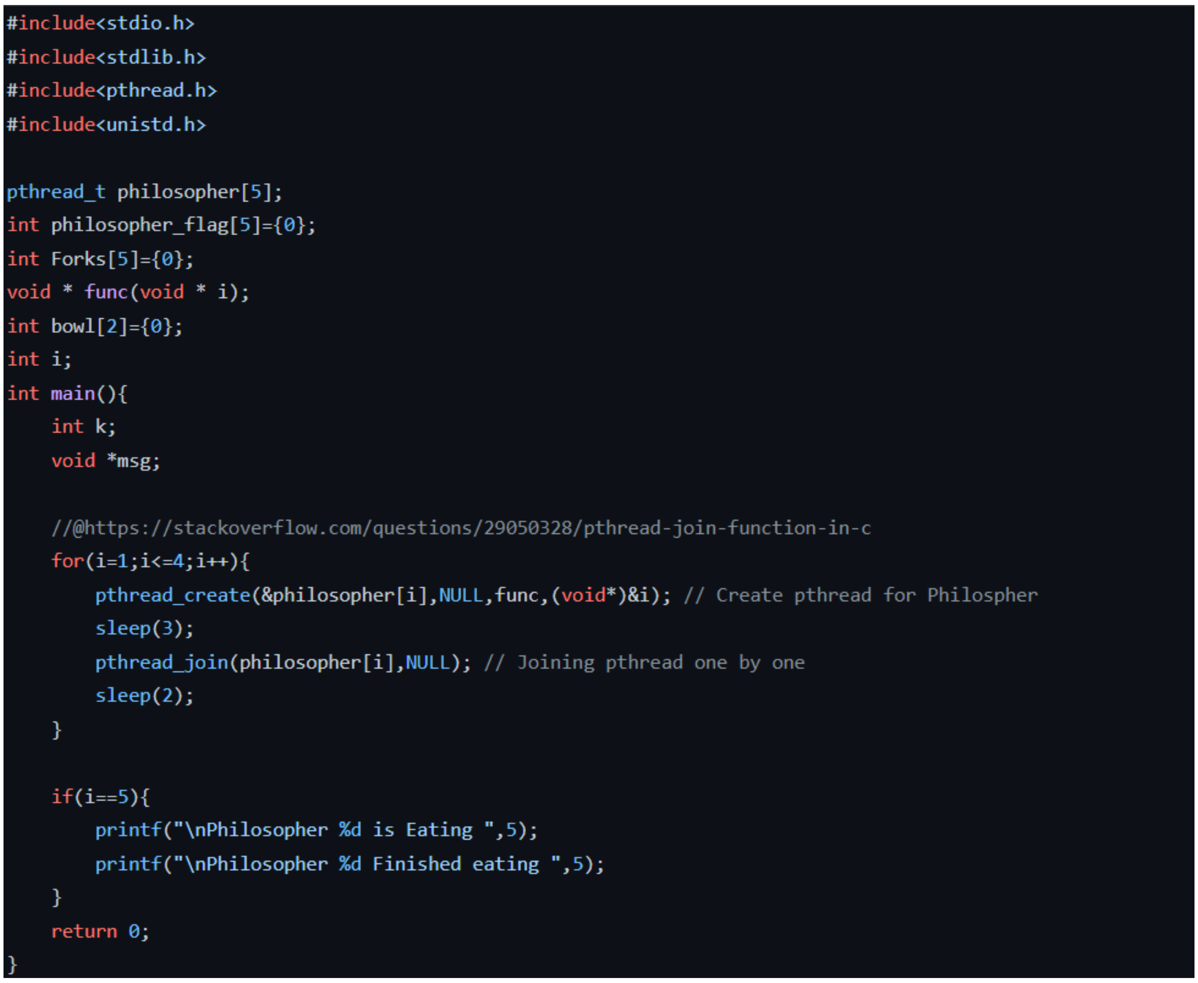
B1:

Without synchronization

Two philosphers can eat at a same time.

Bowl[] array used

Fork[] array used



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

pthread_t philosopher[5];
int philosopher_flag[5]={0};
int Forks[5]={0};
void * func(void * i);
int bowl[2]={0};
int i;
int main(){
    int k;
    void *msg;

    //@https://stackoverflow.com/questions/29050328/pthread-join-function-in-c
    for(i=1;i<=4;i++){
        pthread_create(&philosopher[i],NULL,func,(void*)&i); // Create pthread for Philospher
        sleep(3);
        pthread_join(philosopher[i],NULL); // Joining pthread one by one
        sleep(2);
    }

    if(i==5){
        printf("\nPhilosopher %d is Eating ",5);
        printf("\nPhilosopher %d Finished eating ",5);
    }
    return 0;
}
```

B2:

For synchronization

preventing deadlock Semaphore is used

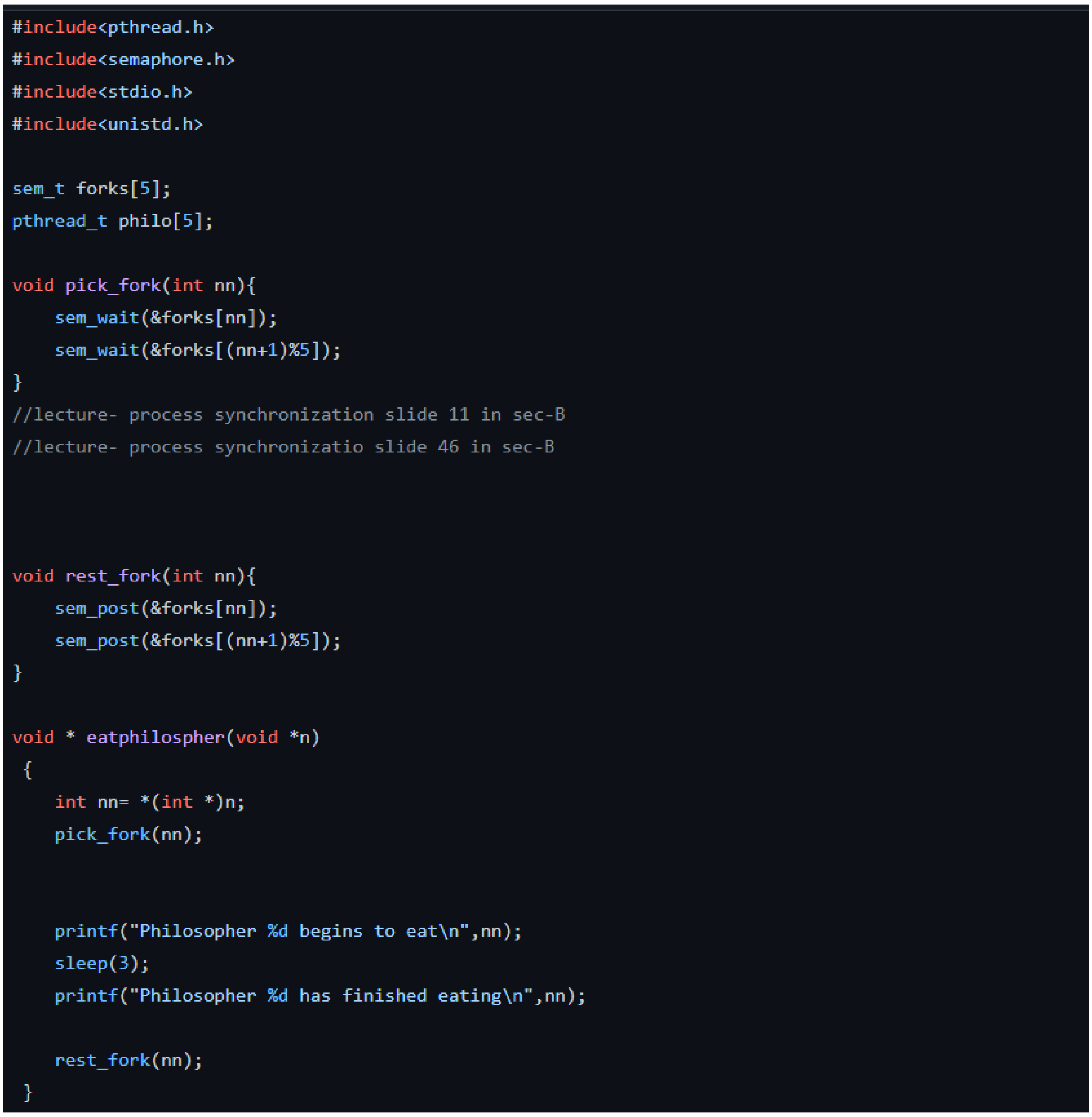
Philosphers are picking forks first from left and right

Forks are in global resources

sem_open() ,sem_unlink(), sem post used

No synchronization used

Two non-adjacent philosphers can eat at same time



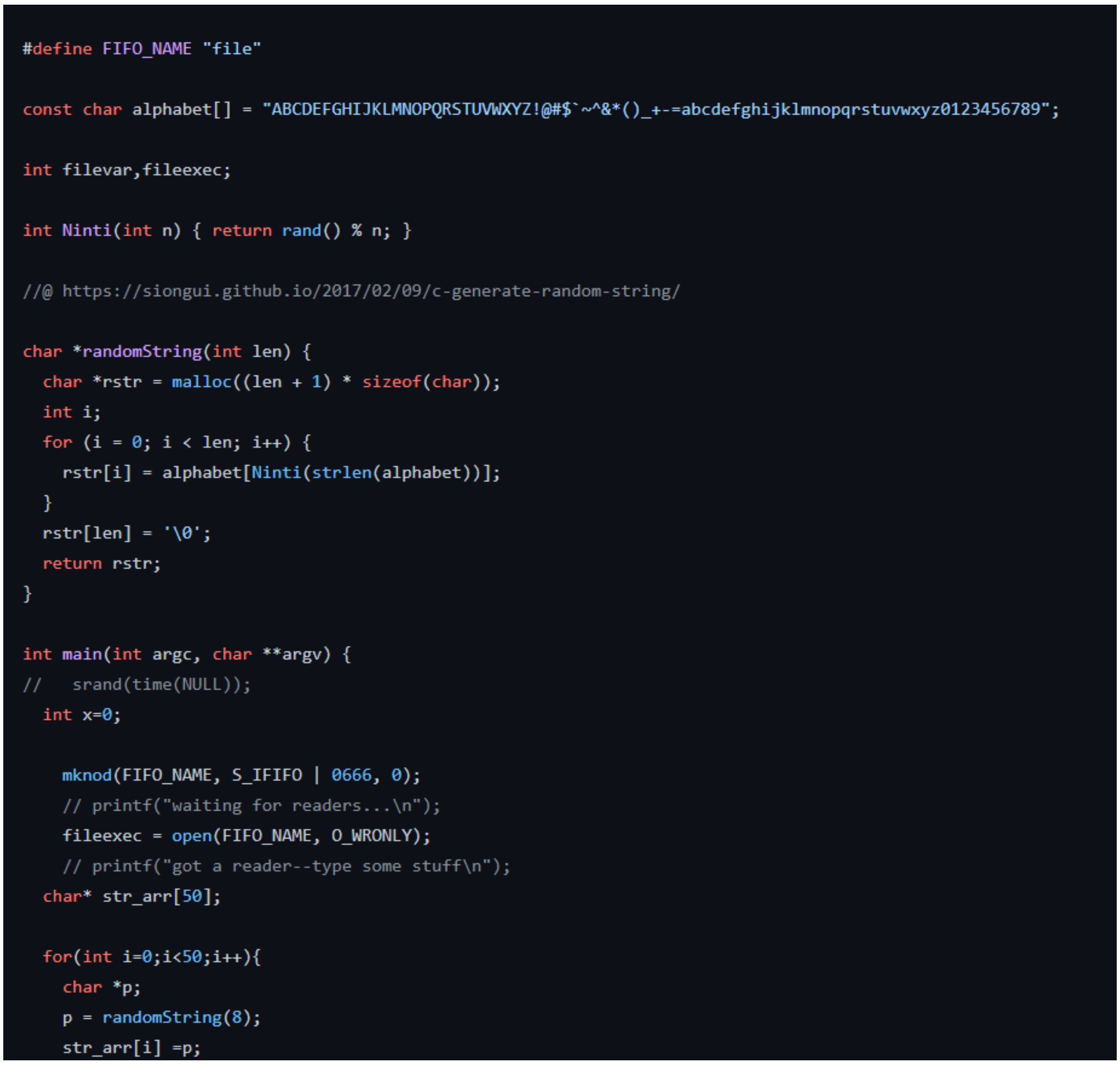
```
#include<pthread.h>
#include<semaphore.h>
#include<stdio.h>
#include<unistd.h>

sem_t forks[5];
pthread_t philo[5];

void pick_fork(int nn){
    sem_wait(&forks[nn]);
    sem_wait(&forks[(nn+1)%5]);
}
//lecture- process synchronization slide 11 in sec-B
//lecture- process synchronizatio slide 46 in sec-B



void rest_fork(int nn){
    sem_post(&forks[nn]);
    sem_post(&forks[(nn+1)%5]);
}

void * eatphilospher(void *n)
 {
    int nn= *(int *)n;
    pick_fork(nn);


    printf("Philosopher %d begins to eat\n",nn);
    sleep(3);
    printf("Philosopher %d has finished eating\n",nn);

    rest_fork(nn);
 }
```

PART-2

FIFO:

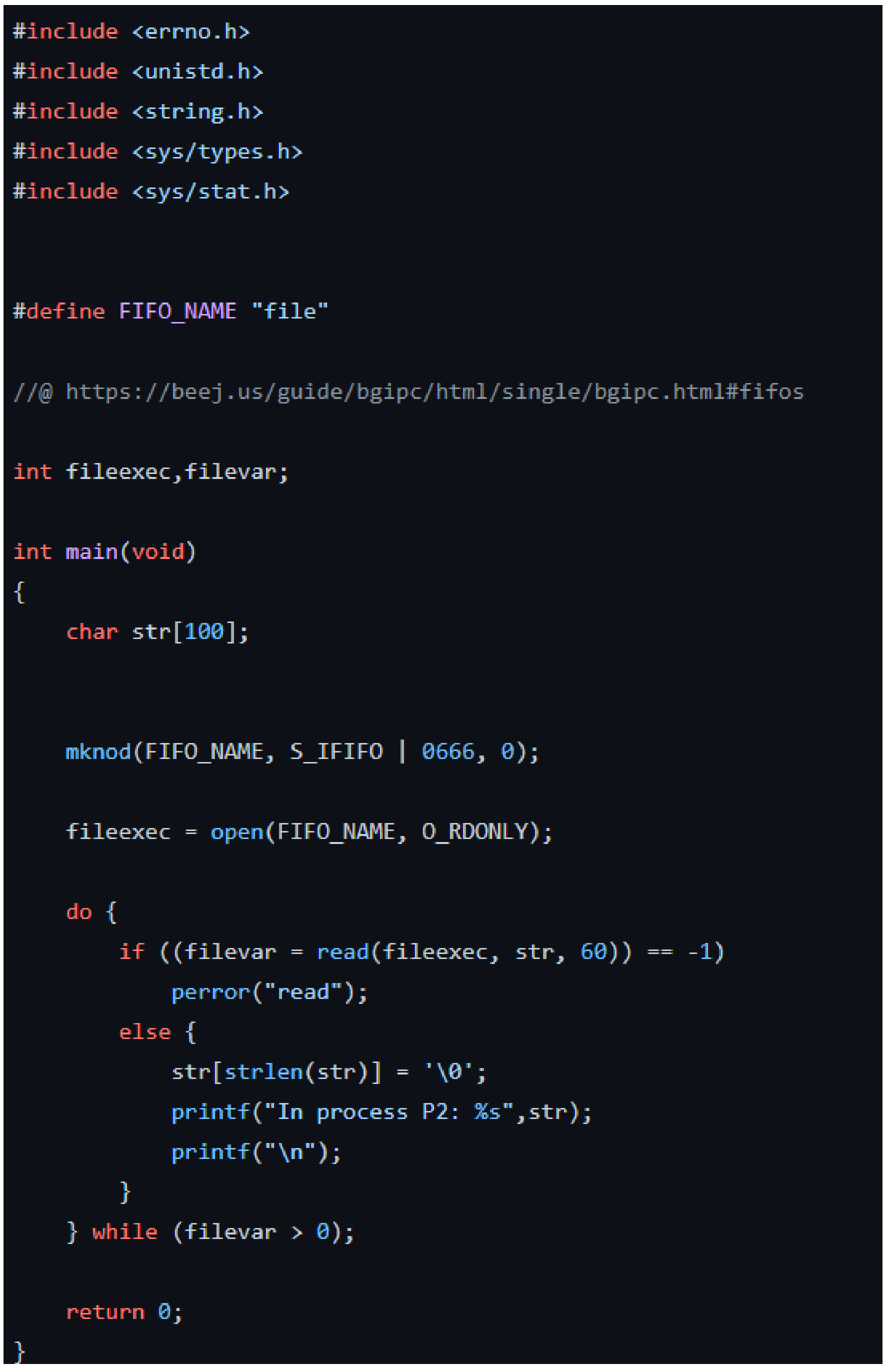
Two processes created speak and tick.

Process P1 writes 50 randomly generated strings of size 8 in an array



```
#define FIFO_NAME "file"

const char alphabet[] = "ABCDEFGHIJKLMNOPQRSTUVWXYZ!@#$`~^&*()_+-=abcdefghijklmnopqrstuvwxyz0123456789";

int filevar,fileexec;

int Ninti(int n) { return rand() % n; }

//@ https://siongui.github.io/2017/02/09/c-generate-random-string/

char *randomString(int len) {
  char *rstr = malloc((len + 1) * sizeof(char));
  int i;
  for (i = 0; i < len; i++) {
    rstr[i] = alphabet[Ninti(strlen(alphabet))];
  }
  rstr[len] = '\0';
  return rstr;
}

int main(int argc, char **argv) {
//   srand(time(NULL));
  int x=0;

    mknod(FIFO_NAME, S_IFIFO | 0666, 0);
    // printf("waiting for readers...\n");
    fileexec = open(FIFO_NAME, O_WRONLY);
    // printf("got a reader--type some stuff\n");
  char* str_arr[50];

  for(int i=0;i<50;i++){
    char *p;
    p = randomString(8);
    str_arr[i] =p;
```

Process P2 reads 5 strings at a time with their index and prints them to the console P1



```c
#include <errno.h>
#include <unistd.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>


#define FIFO_NAME "file"

//@ https://beej.us/guide/bgipc/html/single/bgipc.html#fifos

int fileexec,filevar;

int main(void)
{
    char str[100];


    mknod(FIFO_NAME, S_IFIFO | 0666, 0);

    fileexec = open(FIFO_NAME, O_RDONLY);

    do {
        if ((filevar = read(fileexec, str, 60)) == -1)
            perror("read");
        else {
            str[strlen(str)] = '\0';
            printf("In process P2: %s",str);
            printf("\n");
        }
    } while (filevar > 0);

    return 0;
}
```

SOCKET:

Two processes created client and server

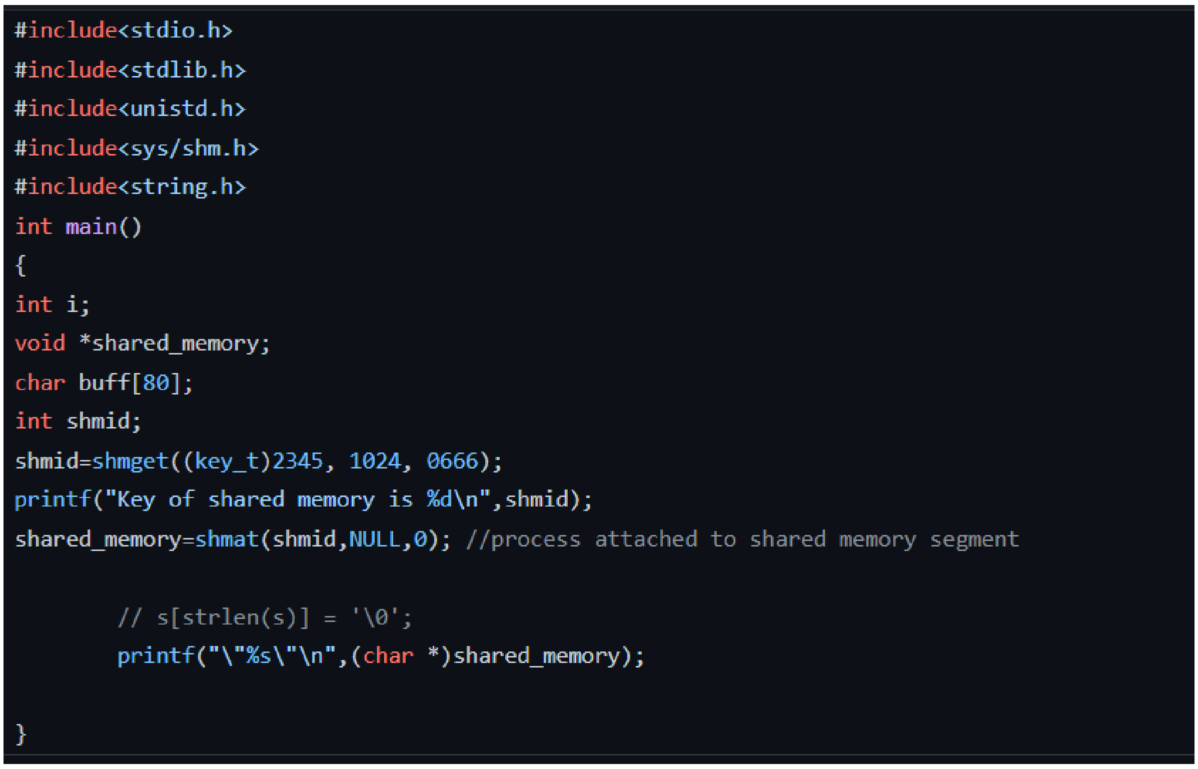
Process P1 writes 50 randomly generated strings of size 8 in an array

```
}


int main(void)
{
  int x=0;

  char* str_arr[50];

  for(int i=0;i<50;i++){
    char *p;
    p = randomString(8);
    str_arr[i] =p;
    // printf("%s ",str_arr[i]);
  }

        serv_addr.sin_family = AF_INET;
        serv_addr.sin_port = htons(PORT);
  int sock = socket(AF_INET, SOCK_STREAM, 0);
  int inet =inet_pton(AF_INET, "127.0.0.1", &serv_addr.sin_addr);
  int client_fileexec = connect(sock, (struct sockaddr*)&serv_addr, sizeof(serv_addr));


    fileexec = open(FILENAME, O_WRONLY);


    for(int i=0;i<10;i++){
    char str[100]="";
    for(int j=0;j<5;j++){
        char result[10];

        strcat(str,str_arr[i+j]);
```

Process P2 reads 5 strings at a time with their index and prints them to the console

```
int main(void)
{
    int server_fileexec = socket(AF_INET, SOCK_STREAM, 0);
    int scokadd= setsockopt(server_fileexec, SOL_SOCKET,SO_REUSEADDR | SO_REUSEPORT, &var,sizeof(var));

    address.sin_family = AF_INET;
    address.sin_addr.s_addr = INADDR_ANY;
    address.sin_port = htons(PORT);

    int bindi= bind(server_fileexec, (struct sockaddr*)&address,sizeof(address));
    int listi= listen(server_fileexec, 3);

    int new_socket = accept(server_fileexec, (struct sockaddr*)&address, (socklen_t*)&addresslen);
    fileexec = open(FILENAME, O_RDONLY);

    char str[100];

    do {
        if ((filevar = read(fileexec, str, 60)) == -1)
            perror("read");
        else {
            str[strlen(str)] = '\0';
            printf("In process P2: %s",str);
            printf("\n");
        }
    } while (filevar > 0);

        close(new_socket);
        shutdown(server_fileexec, SHUT_RDWR);
        return 0;
}
```

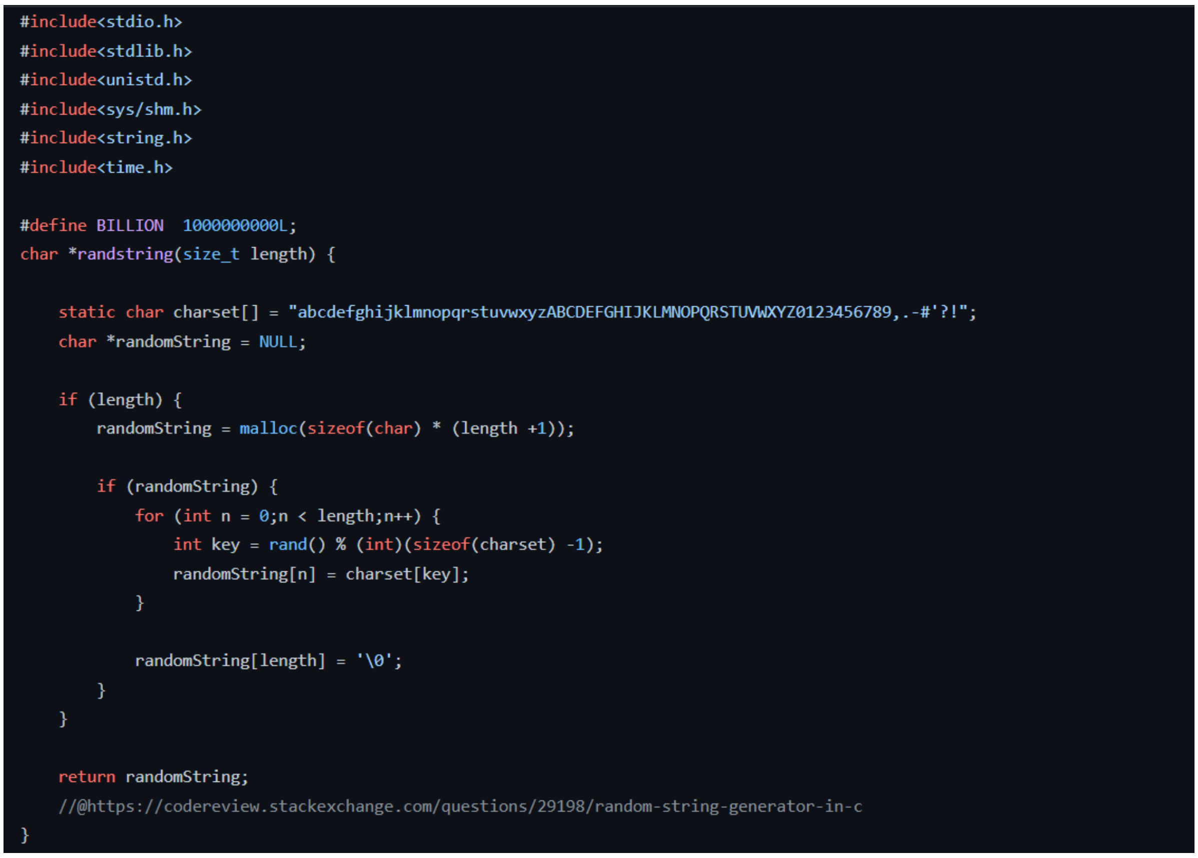
SHARED_MEMORY:



```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<sys/shm.h>
#include<string.h>
int main()
{
int i;
void *shared_memory;
char buff[80];
int shmid;
shmid=shmget((key_t)2345, 1024, 0666);
printf("Key of shared memory is %d\n",shmid);
shared_memory=shmat(shmid,NULL,0); //process attached to shared memory segment

        // s[strlen(s)] = '\0';
        printf("\"%s\"\n",(char *)shared_memory);

}
```

is reading the string and printing out in console.

P2 process is generating 50 random strings of size 5 in sending it to a file further read by P1 process.



```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<sys/shm.h>
#include<string.h>
#include<time.h>

#define BILLION  1000000000L;
char *randstring(size_t length) {

    static char charset[] = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789,.-#'?!";
    char *randomString = NULL;

    if (length) {
        randomString = malloc(sizeof(char) * (length +1));

        if (randomString) {
            for (int n = 0;n < length;n++) {
                int key = rand() % (int)(sizeof(charset) -1);
                randomString[n] = charset[key];
            }

            randomString[length] = '\0';
        }
    }

    return randomString;
    //@https://codereview.stackexchange.com/questions/29198/random-string-generator-in-c
}
```

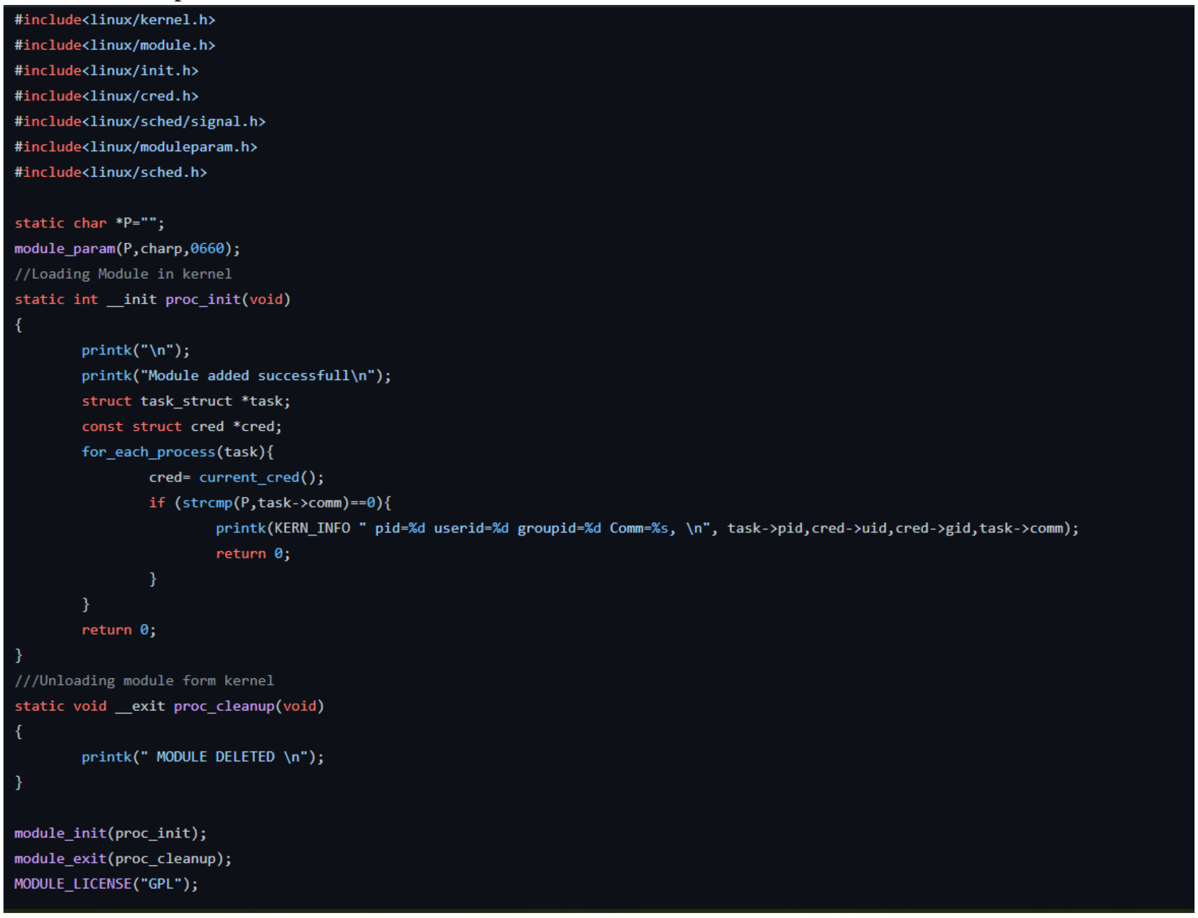
PART-3

Commands:

Insmod mymodule.ko P=<process command>

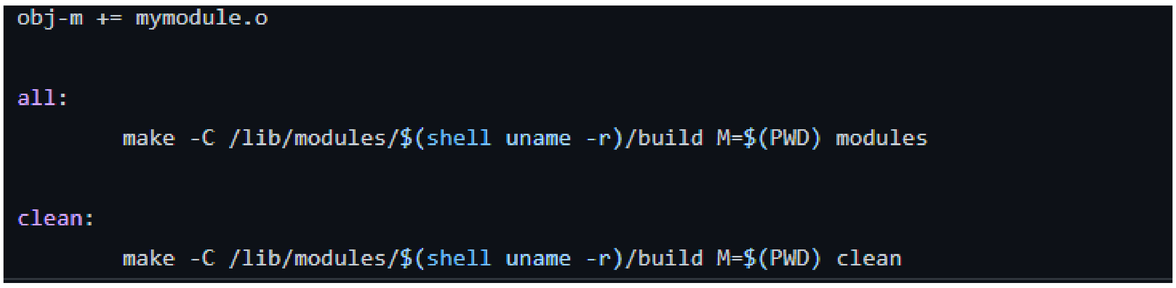
Dmesg

Used: module param



```
#include<linux/kernel.h>
#include<linux/module.h>
#include<linux/init.h>
#include<linux/cred.h>
#include<linux/sched/signal.h>
#include<linux/moduleparam.h>
#include<linux/sched.h>

static char *P="";
module_param(P,charp,0660);
//Loading Module in kernel
static int __init proc_init(void)
{
        printk("\n");
        printk("Module added successfull\n");
        struct task_struct *task;
        const struct cred *cred;
        for_each_process(task){
                cred= current_cred();
                if (strcmp(P,task->comm)==0){
                        printk(KERN_INFO " pid=%d userid=%d groupid=%d Comm=%s, \n", task->pid,cred->uid,cred->gid,task->comm);
                        return 0;
                }
        }
        return 0;
}
///Unloading module form kernel
static void __exit proc_cleanup(void)
{
        printk(" MODULE DELETED \n");
}

module_init(proc_init);
module_exit(proc_cleanup);
MODULE_LICENSE("GPL");
```

Makefile:-



```
obj-m += mymodule.o

all:
        make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules

clean:
        make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
```

Tut-5 of OS