```
HW4

//////////////IPC PIPE//////////////////

#include<stdlib.h>
#include<unistd.h>
#include<stdio.h>
#include<fcntl.h>
#include<string.h>
#include<sys/types.h>
#include<sys/wait.h>

typedef struct{
    char string[50];
    int string_length;
    int USR_Led_Control;
}message;

int flag;

int main()
{
    // from parent to child, parent write, child read
   int pipe_parent2child[2];                                    //0- reading, 1-
writing

   // from child to parent, child write, parent read
   int pipe_child2parent[2];                                    //0- reading, 1-
writing

    pid_t pid;                                                   // pid of child
process
        if (pipe(pipe_parent2child) < 0)                           // create the
pipe
    {
       perror("pipe error");
        printf("pipe_parent2child pipe error\n");
    }    if (pipe(pipe_child2parent) < 0)                          // create the
pipe
    {
       perror("pipe error");
        printf("pipe_child2parent pipe error\n");
    }

    pid = fork();

    if (pid == 0)
    {
flag = 0;
        // child process
        printf("Child process\n");
        message *ptr;
        message mesg_struct;
        ptr = &mesg_struct;

        // child process read
        //close(pipe_child2parent[1]);
        char buff[sizeof(message)] = {0};
        read(pipe_parent2child[0], buff, sizeof(message));
        ptr = (message*)(buff);
        printf("Child Received string: %s, String length = %d, USR led status: %d\n",
ptr->string, ptr->string_length, ptr->USR_Led_Control);
```

```
        close(pipe_parent2child[0]);

        // child process write
        strcpy(mesg_struct.string, "From child to parent msg");
        mesg_struct.string_length = strlen(mesg_struct.string);
        mesg_struct.USR_Led_Control = 0;
        close(pipe_child2parent[0]);
        ptr = &mesg_struct;
        write(pipe_child2parent[1], ptr, sizeof(message));
        printf("Data sent from child to parent\n");
        close(pipe_child2parent[1]);
        exit(1);
    }
    else
    {
flag = 1;
        // parent process
        printf("Parent process\n");

        // parent process write
        message *ptr;
        message mesg_struct;
        ptr = &mesg_struct;
        strcpy(mesg_struct.string, "From parent to child msg");
        mesg_struct.string_length = strlen(mesg_struct.string);
        mesg_struct.USR_Led_Control = 1;

        close(pipe_parent2child[0]); //close read pipe
        write(pipe_parent2child[1], ptr, sizeof(message));
        printf("Data sent from parent to child\n");
        close(pipe_parent2child[1]);
        //open(pipe_parent2child[0]);

        while(!flag);

        // parent process read
        close(pipe_child2parent[1]);
        char buff[sizeof(message)] = {0};
        int ret = read(pipe_child2parent[0], buff, sizeof(message));
        ptr = (message*)(buff);
        printf("Parent Received string: %s, String length = %d, USR led status: %d
\n", ptr->string, ptr->string_length, ptr->USR_Led_Control);
        close(pipe_child2parent[0]);


    }
}

//////////////IPC Shared memory//////////////////

#include<stdlib.h>
#include<unistd.h>
#include<stdio.h>
#include<fcntl.h>
#include<string.h>
#include<sys/shm.h>
#include<sys/stat.h>
#include<sys/mman.h>
#include<semaphore.h>
#include<sys/wait.h>

typedef struct{
```

```
    char string[50];
    int string_length;
    int USR_Led_Control;
}message;

int main()
{
    //size of shared memory object (bytes)
    const int size = sizeof(message);

    //name of shared memory object
    const char *name = "/shared_memory_vp";
    const char *semname = "/my_sem";

    message mesg_struct;
   message *ptr;

    //shared memory file descriptor
    int shm_fd;

    //pointer to shared memory object
    void *mptr;

    //create the shared memory object
    shm_fd = shm_open(name, O_RDWR | O_CREAT , 0666);
    if (shm_fd < 0)
        printf("ERROR shm_open");

    //Config size of share dmemeory
    int ft = ftruncate(shm_fd, size);
    if (ft < 0)
        printf("ERROR ftruncate");

    // mapping of memory segment
    mptr = mmap(NULL, size, PROT_WRITE | PROT_READ, MAP_SHARED, shm_fd, 0);
    if (mptr == NULL)
        printf("ERROR mmap");

    sem_t* sem = sem_open(semname, O_CREAT, 0666, 0);
    if (sem == NULL)
        printf("ERROR sem_open");

        message buff = {0};
        ptr = (char*)&buff;

        strcpy(mesg_struct.string, "From producer to consumer msg");
        mesg_struct.string_length = strlen(mesg_struct.string);
        mesg_struct.USR_Led_Control = 0;
        memcpy(mptr, &mesg_struct, size);

        //unlock the semaphore
        sem_post(sem);

        // locks semaphore
        sem_wait(sem);

        message buff1 = {0};
        ptr = (char*)&buff1;
        memcpy(ptr, (char*)mptr, size);
        printf("Producer string: %s, String length = %d, USR led status: %d\n", ptr-
>string, ptr->string_length, ptr->USR_Led_Control);
```

```
        int shul = shm_unlink(name);
        if (shul < 0)
                printf("ERROR shm_unlink");

    return 0;


}


#include<stdlib.h>
#include<unistd.h>
#include<stdio.h>
#include<fcntl.h>
#include<string.h>
#include<sys/shm.h>
#include<sys/stat.h>
#include<sys/mman.h>
#include<semaphore.h>
#include<sys/wait.h>

typedef struct{
    char string[50];
    int string_length;
    int USR_Led_Control;
}message;

int main()
{
    //size of shared memory object (bytes)
    const int size = sizeof(message);

    //name of shared memory object
    const char *name = "/shared_memory_vp";
    const char *semname = "/my_sem";

    message *ptr;
    message mesg_struct;


    //shared memory file descriptor
    int shm_fd;

    //pointer to shared memory object
    void *mptr;

    //create the shared memory object
    shm_fd = shm_open(name, O_RDWR | O_CREAT , 0666);
    if (shm_fd < 0)
        printf("ERROR shm_open");

    // mapping of memory segment
    mptr = mmap(NULL, size, PROT_WRITE | PROT_READ, MAP_SHARED, shm_fd, 0);
    if (mptr == NULL)
        printf("ERROR mmap");

    sem_t* sem = sem_open(semname, O_CREAT, 0666, 0);
    if (sem == NULL)
        printf("ERROR sem_open");

        // lock semaphore
        sem_wait(sem);
```

```
        message buff = {0};
        ptr = (char*)&buff;
        memcpy(ptr,(char*)mptr, size);
        printf("Consumer string: %s, String length = %d, USR led status: %d\n", ptr-
>string, ptr->string_length, ptr->USR_Led_Control);

        message buff1 = {0};
        ptr = (char*)&buff1;

        strcpy(mesg_struct.string, "From consumer to producer msg");
        mesg_struct.string_length = strlen(mesg_struct.string);
        mesg_struct.USR_Led_Control = 1;

        memcpy(mptr, &mesg_struct, size);

        sem_post(sem);

        close(shm_fd);
        sem_unlink(name);



    return 0;


}


//////////////Kernel module Process tree////////////////////

#include <linux/init.h>
#include <linux/module.h>
#include <linux/moduleparam.h>
#include <linux/errno.h>
#include <linux/sched.h>
#include <linux/pid.h>

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Vipraja Patil");
MODULE_DESCRIPTION("Print Process tree");

static int no_of_children(struct task_struct* child)
{
        int count = 0;
        struct list_head *list  ;
        list_for_each(list, &child->children)

{
                count++;
        }
        return count;
}


static int __init my_module_init(void)
{
    printk(KERN_INFO "my_module_init\n");
    struct task_struct* tsk = current;

 while(tsk->pid != 0)
    {
```

```
        tsk = tsk->parent;
        printk(KERN_INFO "Process name: %s, PID: %d, State: %i, No. of children: %i,
Nice: %d", tsk->comm, tsk->pid, tsk->state, no_of_children(tsk),  task_nice(tsk));

    }

    return 0;
}

static void __exit my_module_exit(void)
{
    printk("Exiting my module\n");
    return;

}

module_init(my_module_init);
module_exit(my_module_exit);

//////////////Kernel module Kthread Problem 4////////////////////

#include <linux/init.h>
#include <linux/module.h>
#include <linux/moduleparam.h>
#include <linux/errno.h>
#include <linux/sched.h>
#include <linux/pid.h>
#include <linux/kernel.h>
#include <linux/timer.h>
#include <linux/kthread.h>
#include <linux/delay.h>
#include <linux/kfifo.h>



MODULE_LICENSE("GPL");
MODULE_AUTHOR("Vipraja Patil");
MODULE_DESCRIPTION("Kthread api");

struct task_struct *p1;
struct task_struct *p2;
struct timer_list myTimer;
#define fifoname "vpfifo"

void firstthread()
{
        kfifo_put(fifoname,current);
        mod_timer(&myTimer, jiffies + msecs_to_jiffies(500));
}

void secondthread()
{
        struct task_struct *data;
        while(1)
        {
        if (kthread_should_stop())
        break;

        int ret = kfifo_get(fifoname, &data);
        if (ret == 0)
        {
                printk("Kfifo empty");
```

```
                //printf("Kfifo empty");
        }
        else
        {
                printk(KERN_LOG "Previous pid: %d, vruntime: %llu",list_prev_entry
(data,tasks)->pid, list_prev_entry(data,tasks)->se.vruntime);
                printk(KERN_LOG "Current pid: %d, vruntime: %llu",data->pid,data-
>vruntime);
                printk(KERN_LOG "Next pid: %d, vruntime: %llu",list_next_entry
(data,tasks)->pid, list_next_entry(data,tasks)->se.vruntime);
        }

        }
}

static int __init my_module_init(void)
{
    printk(KERN_INFO "my_module_init\n");
        INIT_KFIFO(vpfifo);

        // create kthreads
        p1 = kthread_run(firstthread,NULL,"First thread");
        p2 = kthread_run(secondthread, NULL, "Second thread");
        init_timer(&myTimer);
    myTimer.data = (unsigned long)0;
    myTimer.expires = jiffies + msecs_to_jiffies(500);
    myTimer.function = (void(*)(unsigned long))firstthread;
    add_timer(&myTimer);


    return 0;
}

static void __exit my_module_exit(void)
{
    printk("Exiting my module\n");
        kthread_stop(p1);
        kthread_stop(p2);
        del_timer(&myTimer);
    return;

}

module_init(my_module_init);
module_exit(my_module_exit);

//////////////Kernel module Process tree//////////////////

#include <linux/init.h>
#include <linux/module.h>
#include <linux/moduleparam.h>
#include <linux/errno.h>
#include <linux/sched.h>
#include <linux/pid.h>

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Vipraja Patil");
MODULE_DESCRIPTION("Print Process tree");

static int no_of_children(struct task_struct* child)
{
        int count = 0;
```

```
        struct list_head *list  ;
        list_for_each(list, &child->children)

{
                count++;
        }
        return count;
}


static int __init my_module_init(void)
{
    printk(KERN_INFO "my_module_init\n");
    struct task_struct* tsk = current;

 while(tsk->pid != 0)
    {
        tsk = tsk->parent;
        printk(KERN_INFO "Process name: %s, PID: %d, State: %i, No. of children: %i,
Nice: %d", tsk->comm, tsk->pid, tsk->state, no_of_children(tsk),  task_nice(tsk));

    }

    return 0;
}

static void __exit my_module_exit(void)
{
    printk("Exiting my module\n");
    return;

}

module_init(my_module_init);
module_exit(my_module_exit);

//////////////IPC Socket////////////////////

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>

typedef struct{
        char string[50];
        int string_length;
        int USR_Led_Control;
}message;

int main(int argc, char* argv[])
{
        int socket_server,accept_var;
    char buff[256];
    struct sockaddr_in server_addr, client_addr;
        int portno = 3000;
        message *ptr;
    message mesg_struct;
```

```
        socket_server = socket(AF_INET,SOCK_STREAM,0);
        if(!(socket_server))
                printf("ERROR opening socket\n");
        else printf("Successfully created client socket\n");

        server_addr.sin_family = AF_INET;
        struct hostent *host = gethostbyname(argv[1]);
        memcpy(&server_addr.sin_addr, host->h_addr, host->h_length);
        server_addr.sin_port = htons(portno);

        if (connect(socket_server,(struct sockaddr *) &server_addr,sizeof
(server_addr)) < 0)
        printf("ERROR connecting\n");


        //write
    ptr = &mesg_struct;
    strcpy(mesg_struct.string, "client to server");
       mesg_struct.string_length = strlen(mesg_struct.string);
       mesg_struct.USR_Led_Control = 0;
        int send_var = send(socket_server, (void*)(&mesg_struct), sizeof
(mesg_struct),0);
        if (send_var < 0)
         printf("ERROR sending to socket\n");


    int read_var = read(socket_server,buff,sizeof(message));
    if (read_var < 0)
         printf("ERROR reading from socket\n");
         ptr = (message*)(buff);
    printf("Client read string: %s, String length = %d, USR led status: %d\n", ptr-
>string, ptr->string_length, ptr->USR_Led_Control);

    close(socket_server);
    return 0;


}

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <mqueue.h>


typedef struct{
        char string[50];
        int string_length;
        int USR_Led_Control;
}message;

#define SERVER_QUEUE_NAME   "/my_msg_queue_server"
#define QUEUE_PERMISSIONS 0660
#define MAX_MESSAGES 10
#define MAX_MSG_SIZE 256
#define MSG_BUFFER_SIZE MAX_MSG_SIZE + 10

int main()
{
```

```
        mqd_t server, client;   // queue descriptors
    struct mq_attr attr;

        message *ptr;
    message mesg_struct;
    ptr = &mesg_struct;
         char buff[sizeof(message)] = {0};

    attr.mq_flags = 0;
    attr.mq_maxmsg = MAX_MESSAGES;
    attr.mq_msgsize = MAX_MSG_SIZE;
    attr.mq_curmsgs = 0;

        server = mq_open (SERVER_QUEUE_NAME, O_RDWR | O_CREAT, 0666, &attr);
        if (server < 0)
                printf("ERROR opening message queue\n");

        //char* buffrec = (char*)(&mesg_struct);

        //send message to process 2
        strcpy(mesg_struct.string, "From Process 1 to Process 2 msg\n");
    mesg_struct.string_length = strlen(mesg_struct.string);
    mesg_struct.USR_Led_Control = 0;
        char* buffptr = (char*)(&mesg_struct);
        if (mq_send (server, buffptr, sizeof(message), 0) < 0)
                printf("ERROR mq_send\n");
        else printf("Process 1: message sent\n");
        //receive a message from a message queue
        char *buffrec;
        message buffer = {0};
        buffrec = (char*)&buffer;

        if (mq_receive (server, buffrec, sizeof(mesg_struct), 0) < 0)
                printf("ERROR mq_receive\n");
        else printf ("Process 1: message received.\n");

        ptr = (message*)(buffrec);
    printf("Process1 Received string: %s, String length = %d, USR led status: %d\n",
ptr->string, ptr->string_length, ptr->USR_Led_Control);



        mq_close(server);


        return 0;
}



#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>

typedef struct{
        char string[50];
        int string_length;
        int USR_Led_Control;
```

```
}message;

int main()
{
        int socket_server,accept_var;
    char buffer[256];
    struct sockaddr_in server_addr, client_addr;
        int portno = 3000;
        message *ptr;
    message mesg_struct;

        socket_server = socket(AF_INET,SOCK_STREAM,0);
        if(!(socket_server))
                printf("ERROR opening socket\n");
        else printf("Successfully created server socket\n");

        server_addr.sin_family = AF_INET;
    server_addr.sin_addr.s_addr = INADDR_ANY;
    server_addr.sin_port = htons(portno);

        if (bind(socket_server, (struct sockaddr *) &server_addr, sizeof
(server_addr)) < 0)
        printf("ERROR on binding\n");
        else printf("Binding successful\n");

        //listen
        if(listen(socket_server,5) < 0)
                printf("ERROR listening\n");
        else printf("Listening success\n");

        //accept
        accept_var = accept(socket_server, (struct sockaddr *) &client_addr, 0);
     if (accept_var<0)
          printf("ERROR on accept, %i\n", accept_var);

        char buff[sizeof(message)] = {0};

    int read_var = read(accept_var,buff,sizeof(message));

        if (read_var < 0)
                printf("ERROR reading from socket\n");
        ptr = (message*)(buff);
    printf("Server read string: %s, String length = %d, USR led status: %d\n", ptr-
>string, ptr->string_length, ptr->USR_Led_Control);

        ptr = &mesg_struct;
    strcpy(mesg_struct.string, "Server write message\n");
    mesg_struct.string_length = strlen(mesg_struct.string);
    mesg_struct.USR_Led_Control = 1;
    int write_var = write(accept_var,ptr,sizeof(message));
        if (write_var < 0)
                printf("ERROR writing to socket\n");



        close(accept_var);
    close(socket_server);
     return 0;


        return 0;
```

```
}


//////////////IPC Message queue//////////////////
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <mqueue.h>
#include <errno.h>


typedef struct{
        char string[50];
        int string_length;
        int USR_Led_Control;
}message;

#define SERVER_QUEUE_NAME   "/my_msg_queue_server"
#define QUEUE_PERMISSIONS 0666
#define MAX_MESSAGES 10
#define MAX_MSG_SIZE 256
#define MSG_BUFFER_SIZE MAX_MSG_SIZE + 10

int main()
{
    mqd_t server;   // queue descriptors
    struct mq_attr attr;

        message *ptr;
    message mesg_struct;
    ptr = &mesg_struct;
         char buff[sizeof(message)] = {0};

    //attr.mq_flags = 0;
    attr.mq_maxmsg = MAX_MESSAGES;
    attr.mq_msgsize = sizeof(message);
    //attr.mq_curmsgs = 0;

        server = mq_open (SERVER_QUEUE_NAME, O_RDWR | O_CREAT, QUEUE_PERMISSIONS,
&attr);
        if (server < 0)
        printf("ERROR opening message queue\n");

        if (mq_receive (server, buff, sizeof(message), NULL) < 0)
                printf("ERROR mq_receive\n");
        else printf ("Process 2: message received.\n");
        //send message
        char* buffptr = (char*)&mesg_struct;
        strcpy(mesg_struct.string, "From Process 2 to Process 1 msg\n");
    mesg_struct.string_length = strlen(mesg_struct.string);
    mesg_struct.USR_Led_Control = 1;

        if (mq_send (server, buffptr, sizeof(mesg_struct), 0) == -1)
                printf("ERROR mq_send\n");
        else printf("Process 2: message sent\n");


        //receive message
        //char* buffrec = (char*)(&buff);
```

```
        ptr = (message*)(buff);
    printf("Process2 Received string: %s, String length = %d, USR led status: %d\n",
ptr->string, ptr->string_length, ptr->USR_Led_Control);

        mq_unlink(SERVER_QUEUE_NAME);

}

//////////////IPC Shared memory//////////////////

#include<stdlib.h>
#include<unistd.h>
#include<stdio.h>
#include<fcntl.h>
#include<string.h>
#include<sys/shm.h>
#include<sys/stat.h>
#include<sys/mman.h>
#include<semaphore.h>
#include<sys/wait.h>

typedef struct{
    char string[50];
    int string_length;
    int USR_Led_Control;
}message;

int main()
{
    //size of shared memory object (bytes)
    const int size = sizeof(message);

    //name of shared memory object
    const char *name = "/shared_memory_vp";
    const char *semname = "/my_sem";

    message *ptr;
    message mesg_struct;


    //shared memory file descriptor
    int shm_fd;

    //pointer to shared memory object
    void *mptr;

    //create the shared memory object
    shm_fd = shm_open(name, O_RDWR | O_CREAT , 0666);
    if (shm_fd < 0)
        printf("ERROR shm_open");

    // mapping of memory segment
    mptr = mmap(NULL, size, PROT_WRITE | PROT_READ, MAP_SHARED, shm_fd, 0);
    if (mptr == NULL)
        printf("ERROR mmap");

    sem_t* sem = sem_open(semname, O_CREAT, 0666, 0);
    if (sem == NULL)
        printf("ERROR sem_open");

        // lock semaphore
```

```
        sem_wait(sem);

        message buff = {0};
        ptr = (char*)&buff;
        memcpy(ptr,(char*)mptr, size);
        printf("Consumer string: %s, String length = %d, USR led status: %d\n", ptr-
>string, ptr->string_length, ptr->USR_Led_Control);

        message buff1 = {0};
        ptr = (char*)&buff1;

        strcpy(mesg_struct.string, "From consumer to producer msg");
        mesg_struct.string_length = strlen(mesg_struct.string);
        mesg_struct.USR_Led_Control = 1;

        memcpy((char*)mptr, ptr, size);

        sem_post(sem);

        close(shm_fd);
        sem_unlink(name);



    return 0;


}
```