HACETTEPE UNIVERSITY
DEPARTMENT OF COMPUTER ENGINEERING
BBM342 OPERATING SYSTEMS
PROJECT I

Subject : Interprocess Communication, Shared Memory and Semaphores
Submission Date : 24.03.2021
Deadline : 21.04.2021, 23:59 pm
Programming Language : C Programming Language
Advisors : Ahmet Burak CAN, Kayhan İMRE, Feyza Nur Kılıçaslan

1. INTRODUCTION

Operating systems provide a conceptual model consisting of sequential processes running in parallel. Processes can be created and terminated dynamically. Each process has its own address space. Interprocess mechanisms can be used for exchanging data between the incorporating processes. Message Passing Interface (MPI) is one of those standards widely used in parallel programming. In this standard, the processes send and receive messages to exchange data and synchronize with each other.

The aim of this project is to get familiar with both interprocess communication and synchronization mechanisms. In this project, the minimal and simplified set for MPI will be implemented for shared memory multiprocessor architectures. In the minimal set, `MPI_Send` and `MPI_Recv` functions are used for exchanging data. Both send and receive (i.e. Rendezvous, see your lecture notes) are blocking functions. The communication is successful if the source rank is equal to the destination rank and both tag values are the same.

```
int MPI_Init(int *argc, char ***argv);
int MPI_Finalize();
int MPI_Comm_size(int *size);
int MPI_Comm_rank(int *rank);
int MPI_Recv(void *buf, int count, int datatype, int source, int tag);
int MPI_Send(const void *buf, int count, int datatype, int dest, int tag);
```

2. THE PROJECT

In this project, some process management functions are going to be used. To run MPI processes, an application called `my_mpirun` will be developed by you, and its usage will be as follows:

```
[my_computer]$ ./my_mpirun 10 hello
```

In this example, 10 `hello` processes are created (Ranks are ranging from 0 to 9). The source code for `hello` program is given as follows. The processes with even ranks (process identifications for MPI processes) receive and send from/to the next processes. To complement this, the processes with odd ranks send and receive to/from the previous processes.

```
int main(int argc, char *argv[], char *environ[])
{
    int npes,myrank,number;
    MPI_Init(&argc, &argv);
    MPI_Comm_size(&npes);
    MPI_Comm_rank(&myrank);
    if(myrank%2 == 0){
        MPI_Recv(&number,1,sizeof(int),(myrank+1)%npes,0);
        MPI_Send(&number,1,sizeof(int),(myrank+1)%npes,0);
    } else {
        number=myrank;
        MPI_Send(&number,1,sizeof(int),(myrank-1+npes)%npes,0);
        MPI_Recv(&number,1,sizeof(int),(myrank-1+npes)%npes,0);
    }
    MPI_Finalize();
}
```

The code snippet for `my_mpirun` program is given as follows. `my_mpirun` will create the processes and pass some arguments to `hello` processes. Then, `my_mpirun` will wait for those processes to complete their executions.

```
if(pid[count]=fork() == 0){       //fork a new process
    strcpy(code,argv[2]);
    argv[0]=argv[1];              // Set nproc
    sprintf(rank_info,"%d",count);
    argv[1]=rank_info;            // Set rank
    argv[2]=0;
    execve(code,argv,environ);    // Run MPI process
}
```

The implementations of `MPI_Send` and `MPI_Recv` functions are going to be done using shared memory and semaphores. You can use `MPI_Init` and `MPI_Finalize` functions to initialize and destroy things since `MPI_Init` is called once before any other MPI function, and `MPI_Finalize` is called before quitting from the processes. `MPI_Comm_size` and `MPI_Comm_rank` will simply return some information about the processes.

Shared memory between processes can be defined as follows:

```
shm_fd = shm_open("shared_memory", O_CREAT | O_RDWR, 0666);
if(shm_fd < 0)
{
    printf("Error  XYZ1\n");
    exit(0);
}

ftruncate(shm_fd, size);
pointer = mmap(NULL, size, PROT_READ | PROT_WRITE, MAP_SHARED, shm_fd,0);
if( MAP_FAILED  == pointer)
{
    printf("Error XYZ2\n");
    exit(0);
}
```

The semaphores to be used for synchronizing processes can be defined as follows:

```
sprintf(name,"full%d",__rank);
full = sem_open(name, O_CREAT, 0600, 0);
sem_init(full,1,0);

sprintf(name,"empty%d",__rank);
empty = sem_open(name, O_CREAT, 0600, 0);
sem_init(empty,1,BOUNDED_BUFFER_SIZE);
```

3. HOW TO SUBMIT

- Submit your source code directory in a zipped form. Give a README document to explain how to compile the code, and provide a MAKEFILE.
- Write a report (PDF file) that conceptually explains your design. In this report, explain each function from the minimal set by giving pseudocode that shows how data sharing and synchronization are done.
- You work will be graded 30% from your report, 70% from the source code.
- You will use online submission system to submit your projects. (https://submit.cs.hacettepe.edu.tr/) No other submission method (such as CD or email) will be accepted.
- The program must run on DEV (dev.cs.hacettepe.edu.tr) LINUX machine, test your program there.
- Use Piazza for Q&A.

LAST REMARKS:

- Regardless of the length, give **MEANINGFULL** names to your variables, classes and functions.
- Write **READABLE SOURCE CODE** blocks, and use explanatory comments.