Name: Ashmit Thawait

Roll No: 102203790

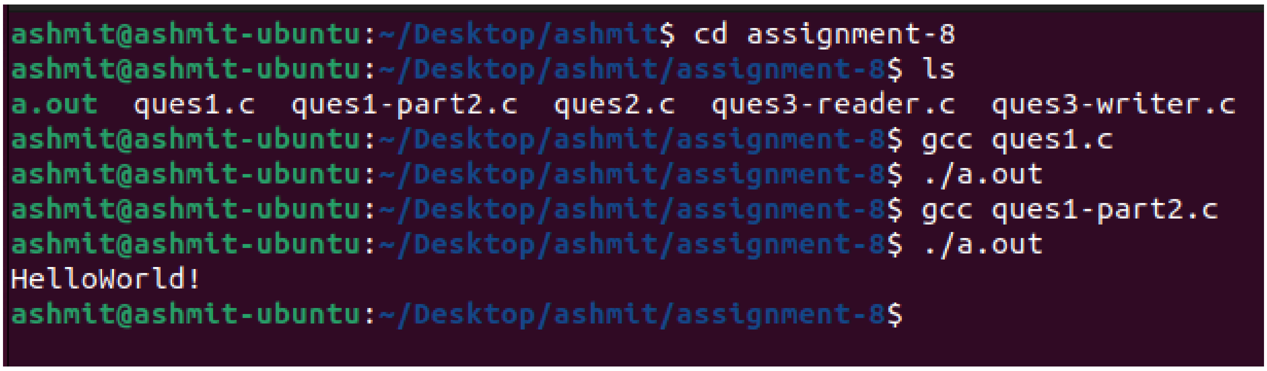
Group: 2CO-17

Lab Assignment 8

Operating Systems (UCS303)

Q1: Write a program to implement a producer-consumer scenario using POSIX shared memory.

Output:



```
ashmit@ashmit-ubuntu:~/Desktop/ashmit$ cd assignment-8
ashmit@ashmit-ubuntu:~/Desktop/ashmit/assignment-8$ ls
a.out  ques1.c  ques1-part2.c  ques2.c  ques3-reader.c  ques3-writer.c
ashmit@ashmit-ubuntu:~/Desktop/ashmit/assignment-8$ gcc ques1.c
ashmit@ashmit-ubuntu:~/Desktop/ashmit/assignment-8$ ./a.out
ashmit@ashmit-ubuntu:~/Desktop/ashmit/assignment-8$ gcc ques1-part2.c
ashmit@ashmit-ubuntu:~/Desktop/ashmit/assignment-8$ ./a.out
HelloWorld!
ashmit@ashmit-ubuntu:~/Desktop/ashmit/assignment-8$
```

CODE:

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <fcntl.h>

#include <sys/shm.h>

#include <sys/stat.h>
```

```
#include <sys/mman.h>
#include <unistd.h>

int main()
{
/* the size (in bytes) of shared memory object */
const int SIZE = 4096;
/* name of the shared memory object */
const char* name = "OS";
/* strings written to shared memory */
const char* message_0 = "Hello";
const char* message_1 = "World!";
/* shared memory file descriptor */
int shm_fd;
/* pointer to shared memory object */
void* ptr;
/* create the shared memory object */
shm_fd = shm_open(name, O_CREAT | O_RDWR, 0666);
/* configure the size of the shared memory object */
ftruncate(shm_fd, SIZE);
/* memory map the shared memory object */
ptr = mmap(0, SIZE, PROT_WRITE, MAP_SHARED, shm_fd, 0);

/* write to the shared memory object */
sprintf(ptr, "%s", message_0);
ptr += strlen(message_0);
sprintf(ptr, "%s", message_1);
ptr += strlen(message_1);
return 0;
}
```

// C program for Consumer process illustrating POSIX shared-memory API.

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <sys/shm.h>
#include <sys/stat.h>
#include <sys/mman.h>
int main()
{
/* the size (in bytes) of shared memory object */
const int SIZE = 4096;
/* name of the shared memory object */
const char* name = "OS";
/* shared memory file descriptor */
int shm_fd;
/* pointer to shared memory object */
void* ptr;
/* open the shared memory object */
```

shm_fd = shm_open(name, O_RDONLY, 0666);

/* memory map the shared memory object */

ptr = mmap(0, SIZE, PROT_READ, MAP_SHARED, shm_fd, 0);

/* read from the shared memory object */

printf("%s", (char*)ptr);

/* remove the shared memory object */

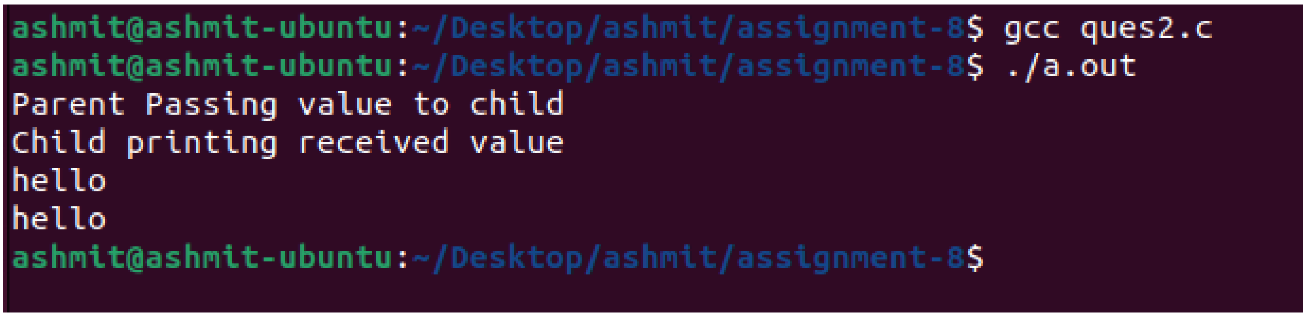
shm_unlink(name);

return 0;

}

Q2: Write a program using Pipes to implement Inter-Process Communication between the parent and child processes.

Output:



```
ashmit@ashmit-ubuntu:~/Desktop/ashmit/assignment-8$ gcc ques2.c
ashmit@ashmit-ubuntu:~/Desktop/ashmit/assignment-8$ ./a.out
Parent Passing value to child
Child printing received value
hello
hello
ashmit@ashmit-ubuntu:~/Desktop/ashmit/assignment-8$
```

Code:

#include<stdio.h>

#include<unistd.h>

#include<sys/types.h>

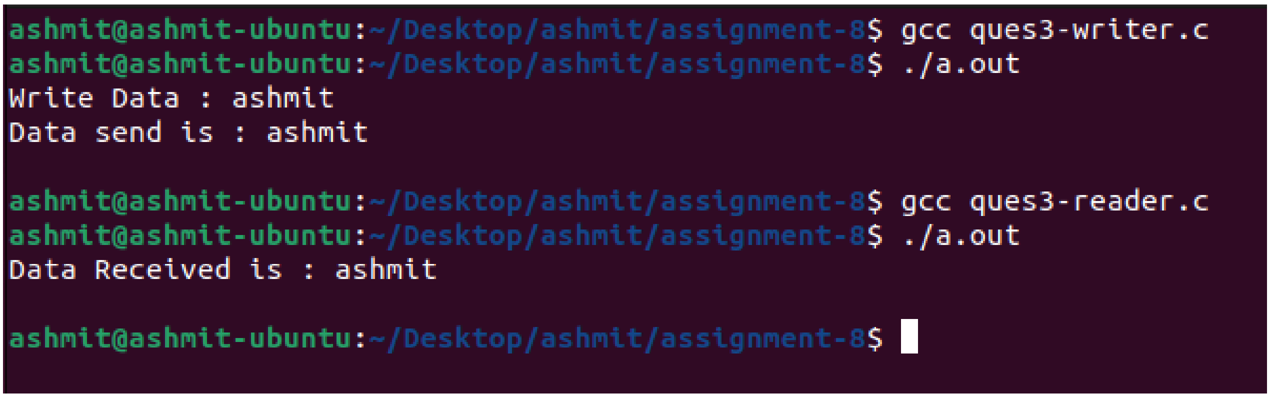
#include<sys/wait.h>

int main()

```
{
int fd[2],n;
char buffer[100];
pid_t p;
pipe(fd); //creates a unidirectional pipe with two end fd[0] and fd[1]
p=fork();
if(p>0) //parent
{
printf("Parent Passing value to child\n");
write(fd[1],"hello\n",6); //fd[1] is the write end of the pipe
sleep(3);
}
else // child
{
printf("Child printing received value\n");
n=read(fd[0],buffer,100); //fd[0] is the read end of the pipe
printf("%s",buffer);
write(1,buffer,n);
}
}
```

O3: Write a program to implement IPC through message queues.

Output:



```
ashmit@ashmit-ubuntu:~/Desktop/ashmit/assignment-8$ gcc ques3-writer.c
ashmit@ashmit-ubuntu:~/Desktop/ashmit/assignment-8$ ./a.out
Write Data : ashmit
Data send is : ashmit

ashmit@ashmit-ubuntu:~/Desktop/ashmit/assignment-8$ gcc ques3-reader.c
ashmit@ashmit-ubuntu:~/Desktop/ashmit/assignment-8$ ./a.out
Data Received is : ashmit

ashmit@ashmit-ubuntu:~/Desktop/ashmit/assignment-8$ 
```

Code:

// C Program for Message Queue (Writer Process)

```
#include <stdio.h>

#include <sys/ipc.h>

#include <sys/msg.h>

#define MAX 10

// structure for message queue

struct mesg_buffer {

long mesg_type;

char mesg_text[100];

} message;

int main()

{

key_t key;

int msgid;

// ftok to generate unique key
key = ftok("progfile", 65);
// msgget creates a message queue
// and returns identifier
msgid = msgget(key, 0666 | IPC_CREAT);
message.mesg_type = 1;
printf("Write Data : ");
fgets(message.mesg_text,MAX,stdin);
// msgsnd to send message
msgsnd(msgid, &message, sizeof(message), 0);
// display the message
printf("Data send is : %s \n", message.mesg_text);
return 0;
}
```

// C Program for Message Queue (Reader Process)

```
#include <stdio.h>
#include <sys/ipc.h>
#include <sys/msg.h>
// structure for message queue
struct mesg_buffer {
long mesg_type;
char mesg_text[100];
} message;
int main()
```

```
{
key_t key;
int msgid;
// ftok to generate unique key
key = ftok("progfile", 65);

// msgget creates a message queue
// and returns identifier
msgid = msgget(key, 0666 | IPC_CREAT);

// msgrcv to receive message
msgrcv(msgid, &message, sizeof(message), 1, 0);

// display the message
printf("Data Received is : %s \n", message.mesg_text);

// to destroy the message queue
msgctl(msgid, IPC_RMID, NULL);
return 0;
}
```