Department of Computer Science and Engineering
Institute of Technical Education & Research, SOA, Deemed to be University

MINOR ASSIGNMENT-11

Inter-Process Communication: Shared Memory & Message Queue

Practical Programming with C (CSE 3544)

Publish on: 21-12-2024 **Submission on:** 28-12-2024
Course Outcome: CO_5 **Program Outcome:** PO_3 **Learning Level:** L_5

Problem Statement:

Experiment with inter-process communication mechanism(IPC) using shared memory and message queue..

Assignment Objectives:

Students will be able to learn how communication between processes is established using the IPC mechanisms, shared memory and message queue.

Answer the followings:

1. Write a program to create a shared memory segment of size 10 bytes. Make 4 attachments to the shared memory segment to the address space of the calling process and print the number of attachments using the structure filed *number of current attachments* present in the structure **`shmid_ds`** defined in the header **`<sys/shm.h>`**. Check the number of attachment using the shell provided command **`ipcs -m`**.

Code here **Specify: input & output**

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

#define SHM_SIZE 10

int main() {
   int shmid = shmget(IPC_PRIVATE, SHM_SIZE, IPC_CREAT | 0666);
   if (shmid == -1) {
      perror("shmget");
      exit(1);
   }

   void *shmaddrs[4];
   for (int i = 0; i < 4; i++) {
      shmaddrs[i] = shmat(shmid, NULL, 0);
      if (shmaddrs[i] == (void*) -1) {
         perror("shmat");
         exit(1);
      }
   }

   struct shmid_ds shm_info;
   if (shmctl(shmid, IPC_STAT, &shm_info) == -1) {
      perror("shmctl");
      exit(1);
   }
   printf("Number of attachments: %ld\n", shm_info.shm_nattch);

for (int i = 0; i < 4; i++) {
      if (shmdt(shmaddrs[i]) == -1) {
         perror("shmdt");
         exit(1);
      }
   }

   if (shmctl(shmid, IPC_RMID, NULL) == -1) {
      perror("shmctl");
      exit(1);
   }

   return 0;
}
```

2. Create a C code named **shmwriter.c** to create a shared memory segment of integer size and store 500 to the segment. Create another program named **shmreader.c** to access the stored value from the shared memory segment and display it. Let the **shmreader.c** update the value to 600. Now update the **shmwriter.c** code to get the updated value and display it. You are not allowed to use `semaphore`.

Code here **Specify: input & output**

shmwriter.c

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

#define SHM_SIZE sizeof(int)

int main() {
   key_t key = ftok("shmfile", 65);
   int shmid = shmget(key, SHM_SIZE, 0666|IPC_CREAT);
   if (shmid == -1) {
      perror("shmget");
      exit(1);
   }

   int *data = (int*) shmat(shmid, NULL, 0);
   if (data == (int*) -1) {
      perror("shmat");
      exit(1);
   }

   *data = 500;
   printf("Initial value written to shared memory: %d\n", *data);

   shmdt(data);
   return 0;
}
```

shmreader.c

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

#define SHM_SIZE sizeof(int)

int main() {
   key_t key = ftok("shmfile", 65);
   int shmid = shmget(key, SHM_SIZE, 0666);
   if (shmid == -1) {
      perror("shmget");
      exit(1);
   }

   int *data = (int*) shmat(shmid, NULL, 0);
   if (data == (int*) -1) {
      perror("shmat");
      exit(1);
   }

   printf("Value read from shared memory: %d\n", *data);
   *data = 600;
   printf("Updated value written to shared memory: %d\n", *data);

   shmdt(data);
   return 0;
}
```

Updating shmwriter.c

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

#define SHM_SIZE sizeof(int)

int main() {
   key_t key = ftok("shmfile", 65);
   int shmid = shmget(key, SHM_SIZE, 0666);
   if (shmid == -1) {
      perror("shmget");
      exit(1);
   }

   int *data = (int*) shmat(shmid, NULL, 0);
   if (data == (int*) -1) {
      perror("shmat");
      exit(1);
   }

   printf("Value read from shared memory after update: %d\n", *data);

   shmdt(data);
   return 0;
}
```

3. Create 2 processes using **`fork()`**. The child will sent a number to parent using shared memory segment. The parent willl display the received number and doubles it and sends back to the client. The client will display the received number.

Code here **Specify: input & output**

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/wait.h>

#define SHM_SIZE sizeof(int)

int main() {
   key_t key = ftok("shmfile", 65);
   int shmid = shmget(key, SHM_SIZE, 0666|IPC_CREAT);
   if (shmid == -1) {
      perror("shmget");
      exit(1);
   }

   pid_t pid = fork();

   if (pid < 0) {
      perror("fork");
      exit(1);
   } else if (pid == 0) {
      int *shared_data = (int*) shmat(shmid, NULL, 0);
      if (shared_data == (int*) -1) {
         perror("shmat");
         exit(1);
      }
      *shared_data = 100;
      printf("Child: Sent %d to parent\n", *shared_data);
      sleep(1);
      printf("Child: Received %d from parent\n", *shared_data);
      shmdt(shared_data);
      exit(0);
   } else {
      wait(NULL);
      int *shared_data = (int*) shmat(shmid, NULL, 0);
      if (shared_data == (int*) -1) {
         perror("shmat");
         exit(1);
      }
      printf("Parent: Received %d from child\n", *shared_data);
      *shared_data *= 2;
      shmdt(shared_data);
      wait(NULL);
      shmctl(shmid, IPC_RMID, NULL);
   }

   return 0;
}
```

4. Design a C code to create a `message queue` and add 4 messages to the queue. Create a receiver code to receive all the messages from the queue till the queue is empty.

Code here **Specify: input & output**

msgsender.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/ipc.h>
#include <sys/msg.h>

#define MAX 100

struct msg_buffer {
   long msg_type;
   char msg_text[MAX];
} message;

int main() {
   key_t key;
   int msgid;

   key = ftok("msgqueuefile", 65);

   msgid = msgget(key, 0666 | IPC_CREAT);
   if (msgid == -1) {
      perror("msgget");
      exit(1);
   }

   for (int i = 1; i <= 4; i++) {
      message.msg_type = 1;
      snprintf(message.msg_text, MAX, "Message %d", i);
      if (msgsnd(msgid, &message, sizeof(message.msg_text),
0) == -1) {
         perror("msgsnd");
         exit(1);
      }
      printf("Sent: %s\n", message.msg_text);
   }
   return 0;
}
```

msgreceiver.c

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/ipc.h>
#include <sys/msg.h>

#define MAX 100

struct msg_buffer {
   long msg_type;
   char msg_text[MAX];
} message;

int main() {
   key_t key;
   int msgid;

   key = ftok("msgqueuefile", 65);

   msgid = msgget(key, 0666 | IPC_CREAT);
   if (msgid == -1) {
      perror("msgget");
      exit(1);
   }

   while (msgrcv(msgid, &message,
sizeof(message.msg_text), 1, 0) != -1) {
      printf("Received: %s\n", message.msg_text);
   }

   if (msgrcv(msgid, &message, sizeof(message.msg_text), 1,
IPC_NOWAIT) == -1) {
      perror("msgrcv");
   }

   if (msgctl(msgid, IPC_RMID, NULL) == -1) {
      perror("msgctl");
      exit(1);
   }
   return 0;
}
```

5∗. Write a C code to create a message queue. Write 6 messages of message the type 10, 30, 46, 67, 78, and 88 onto the queue. Create a receiver code to receive the message depending on the **`msgtyp`** parameter of the **`msgrcv`** system call as **`msgtyp=-10`**, **`msgtyp=100`**, **`msgtyp=-46`**, **`msgtyp=0`**, and **`msgtyp=88`** respectively.

Code here **Specify: input & output**

msgsender.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/ipc.h>
#include <sys/msg.h>

#define MAX 100

struct msg_buffer {
   long msg_type;
   char msg_text[MAX];
} message;

int main() {
   key_t key;
   int msgid;

   key = ftok("msgqueuefile", 65);
   msgid = msgget(key, 0666 | IPC_CREAT);
   if (msgid == -1) {
      perror("msgget");
      exit(1);
   }

   long msg_types[] = {10, 30, 46, 67, 78, 88};
   for (int i = 0; i < 6; i++) {
      message.msg_type = msg_types[i];
      snprintf(message.msg_text, MAX, "Message of type %ld",
msg_types[i]);
      if (msgsnd(msgid, &message, sizeof(message.msg_text), 0)
== -1) {
         perror("msgsnd");
         exit(1);
      }
      printf("Sent: %s\n", message.msg_text);
   }

   return 0;
}
```

msgreceiver.c

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/ipc.h>
#include <sys/msg.h>

#define MAX 100

struct msg_buffer {
   long msg_type;
   char msg_text[MAX];
} message;

void receive_message(int msgid, long msgtyp) {
   if (msgrcv(msgid, &message, sizeof(message.msg_text),
msgtyp, 0) != -1) {
      printf("Received: %s\n", message.msg_text);
   } else {
      perror("msgrcv");
   }
}

int main() {
   key_t key;
   int msgid;

   key = ftok("msgqueuefile", 65);
   msgid = msgget(key, 0666 | IPC_CREAT);
   if (msgid == -1) {
      perror("msgget");
      exit(1);
   }

   long msg_types[] = {-10, 100, -46, 0, 88};
   for (int i = 0; i < 5; i++) {
      receive_message(msgid, msg_types[i]);
   }

   if (msgctl(msgid, IPC_RMID, NULL) == -1) {
      perror("msgctl");
      exit(1);
   }

   return 0;
}
```

6∗. Write a program to read a string **iter** and encrypt the string using a cryptographic technique called `caeser` cipher with a key value of 5. The encryption can be represented using modular arithmetic by first transforming the letters into numbers, according to the scheme, $A = 0$, $B = 1$,..., $Z = 25$. Encryption of a letter say x by a key k can be described mathematically as $E_k(x) = (x + k)$ mod 26. After encryption write the key value and encrypted message on to the queue. Create a receiver code to get the message and the key value. Decrypt the received message using the reverse process as $D_k(x) = (x - k)$ mod 26. After decryption display the message on the receiver side.

```
Example-1

Text :  asdzf
key:    3
Cipher: dvgci


Example-2

Text :  ATTACKATONCE
Shift:  4
Cipher: EXXEGOEXSRGI
```

Code here **Specify: input & output**

msgsender.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/ipc.h>
#include <sys/msg.h>

#define MAX 100

struct msg_buffer {
   long msg_type;
   char msg_text[MAX];
} message;

int main() {
   key_t key;
   int msgid;

   key = ftok("msgqueuefile", 65);
   msgid = msgget(key, 0666 | IPC_CREAT);
   if (msgid == -1) {
      perror("msgget");
      exit(1);
   }

   long msg_types[] = {10, 30, 46, 67, 78, 88};
   for (int i = 0; i < 6; i++) {
      message.msg_type = msg_types[i];
      snprintf(message.msg_text, MAX, "Message of type %ld", msg_types[i]);
      if (msgsnd(msgid, &message, sizeof(message.msg_text), 0) == -1) {
         perror("msgsnd");
         exit(1);
      }
      printf("Sent: %s\n", message.msg_text);
   }

   return 0;
}
```

Code here **Specify: input & output**

msgreceiver.c

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/ipc.h>
#include <sys/msg.h>

#define MAX 100

struct msg_buffer {
   long msg_type;
   char msg_text[MAX];
} message;

void receive_message(int msgid, long msgtyp) {
   if (msgrcv(msgid, &message, sizeof(message.msg_text), msgtyp, 0) != -1) {
      printf("Received: %s\n", message.msg_text);
   } else {
      perror("msgrcv");
   }
}

int main() {
   key_t key;
   int msgid;

   key = ftok("msgqueuefile", 65);
   msgid = msgget(key, 0666 | IPC_CREAT);
   if (msgid == -1) {
      perror("msgget");
      exit(1);
   }

   long msg_types[] = {-10, 100, -46, 0, 88};
   for (int i = 0; i < 5; i++) {
      receive_message(msgid, msg_types[i]);
   }

   if (msgctl(msgid, IPC_RMID, NULL) == -1) {
      perror("msgctl");
      exit(1);
   }

   return 0;
}
```