Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	[@KLWKS_BOT THANOS]

Experiment Title: Inter-Process Communication

Aim/Objective: Students should be able to understand the concepts of Inter-process Communication, and learns related to pipes, shared memory, semaphores, and signals. Works on the problems related to inter-process communication.

Description:

Inter-Process Communication is a type of mechanism usually provided by the operating system (or OS). The main aim or goal of this mechanism is to provide communication in between several processes. In short, the intercommunication allows a process to let another process know that some event has occurred.

Prerequisite:

- **Basic functionality of IPC.**
- **Complete idea of different methods like shmget, and segptr.**
- **Basicfunctionality of threading and synchronization concepts.**

Pre-Lab Task:

IPC terms	FUNCTIONALITY
Pipes	Data flow between processes, typically within the same system.
Shared memory	Allows processes to access common memory space for communication.

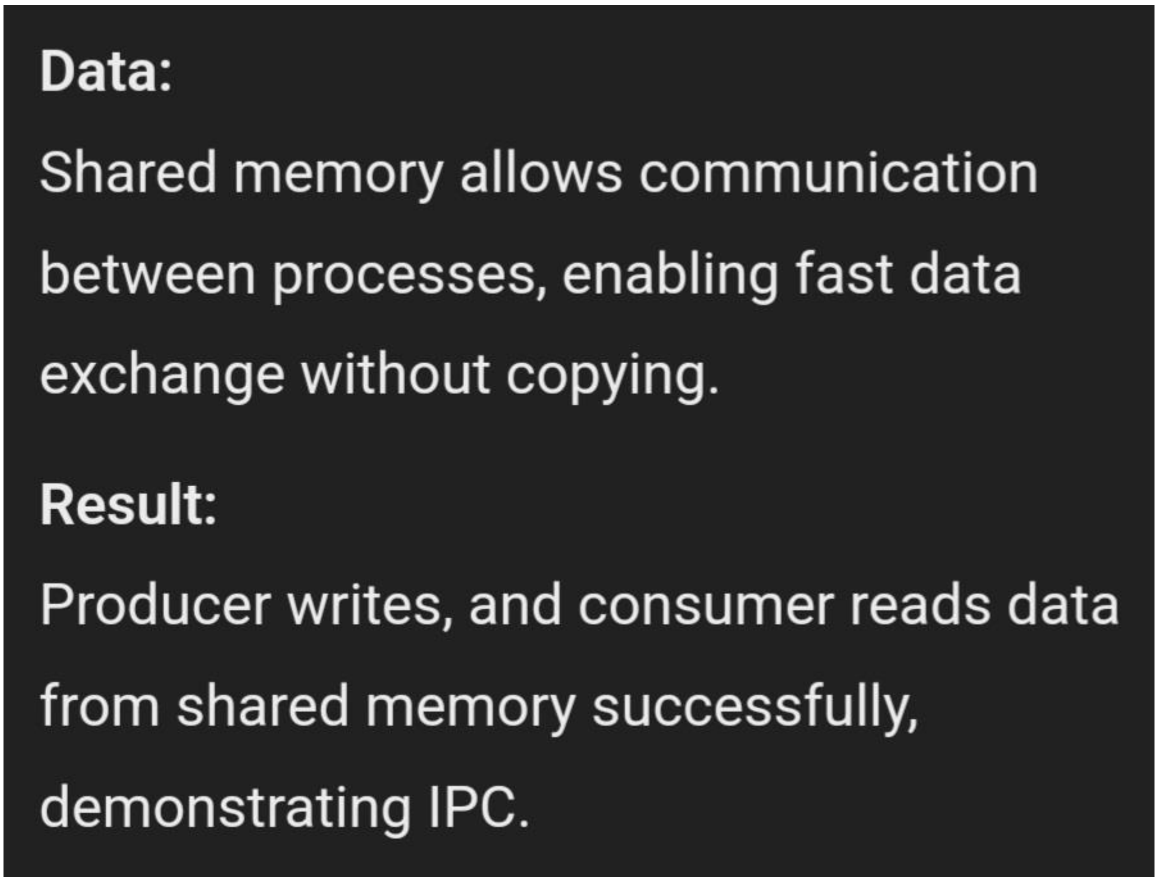
IPC terms	FUNCTIONALITY
Semaphores	Synchronization tools for controlling access to shared resources.
Signals	Software interrupts used for process communication or notification.
Sockets	Endpoints for communication between processes, often over a network.
Message Queues	Queue structure for asynchronous message passing between processes.

In lab task

1. Write a C program to implement IPC using shared memory.

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

#define SHM_SIZE 1024

int main() {
    int shmid;
    key_t key;
    char *shm, *s;

    key = ftok(".", 'S');
    shmid = shmget(key, SHM_SIZE, IPC_CREAT | 0666);
    if (shmid == -1) {
        perror("shmget");
        exit(1);
    }

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

    s = shm;
    char *message = "Hello, Shared Memory!";
    strcpy(s, message);

    shmdt(shm);

    pid_t pid = fork();
    if (pid == -1) {
        perror("fork");
        exit(1);
    } else if (pid == 0) {
        shm = shmat(shmid, NULL, 0);
        if (shm == (char *)-1) {
            perror("shmat (child)");
            exit(1);
}
        s = shm;
        printf("Consumer: Received data from shared memory: %s\n", s);
        shmdt(shm);
    } else {
        wait(NULL);
        shmctl(shmid, IPC_RMID, NULL);
    }

    return 0;
}
```

Data and Results



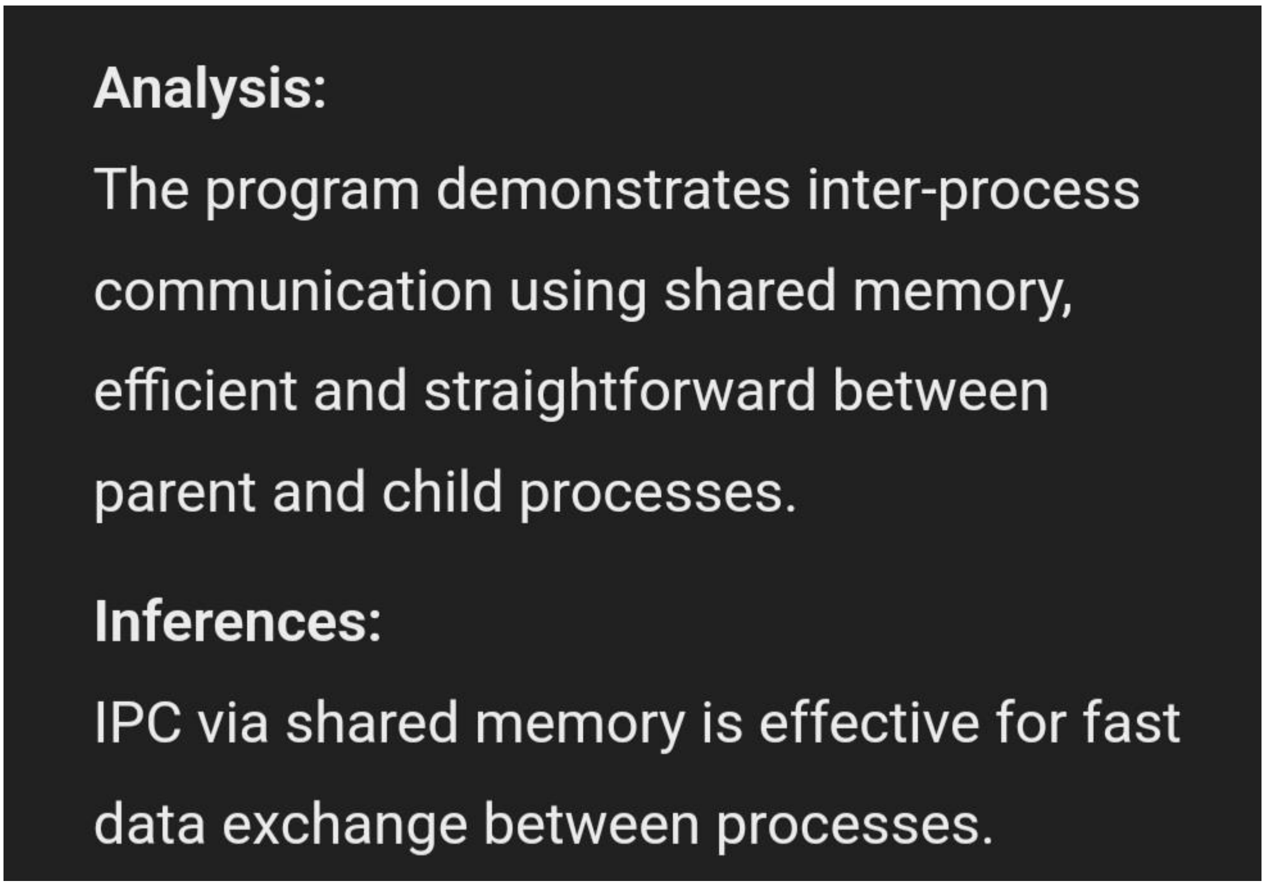
Data:

Shared memory allows communication between processes, enabling fast data exchange without copying.

Result:

Producer writes, and consumer reads data from shared memory successfully, demonstrating IPC.

Analysis and Inferences



Analysis:

The program demonstrates inter-process communication using shared memory, efficient and straightforward between parent and child processes.

Inferences:

IPC via shared memory is effective for fast data exchange between processes.

Post Lab Task:

1. Write a C program to print numbers in a sequence using Thread Synchronization

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

#define NUM_THREADS 3
#define MAX_COUNT 10

int count = 0;
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t cond = PTHREAD_COND_INITIALIZER;

void *printOdd(void *arg) {
   for (;;) {
      pthread_mutex_lock(&mutex);
      while (count % 2 == 0 && count < MAX_COUNT) {
         pthread_cond_wait(&cond, &mutex);
      }
      if (count >= MAX_COUNT) {
         pthread_mutex_unlock(&mutex);
         pthread_exit(NULL);
      }
      printf("Odd: %d\n", count);
      count++;
      pthread_cond_signal(&cond);
      pthread_mutex_unlock(&mutex);
   }
   return NULL;
}

void *printEven(void *arg) {
   for (;;) {
      pthread_mutex_lock(&mutex);
      while (count % 2 != 0 && count < MAX_COUNT) {
```

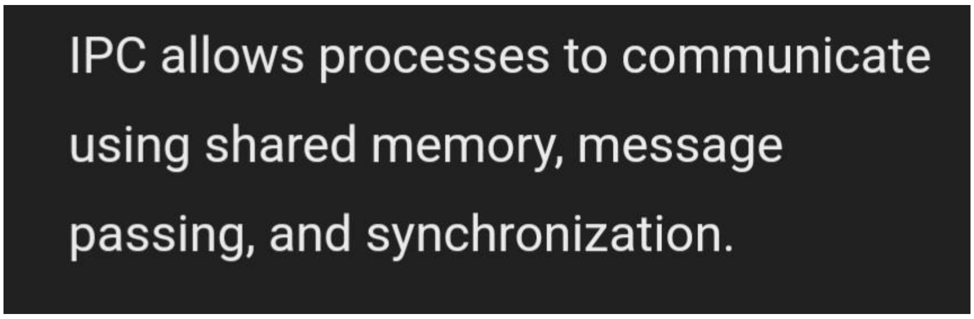
```
  pthread_cond_wait(&cond, &mutex);
     }
     if (count >= MAX_COUNT) {
        pthread_mutex_unlock(&mutex);
        pthread_exit(NULL);
     }
     printf("Even: %d\n", count);
     count++;
     pthread_cond_signal(&cond);
     pthread_mutex_unlock(&mutex);
  }
  return NULL;
}

int main() {
  pthread_t threads[NUM_THREADS];
  pthread_create(&threads[0], NULL, printOdd, NULL);
  pthread_create(&threads[1], NULL, printEven, NULL);

  for (int i = 0; i < NUM_THREADS; i++) {
     pthread_join(threads[i], NULL);
  }

  pthread_mutex_destroy(&mutex);
  pthread_cond_destroy(&cond);

  return 0;
}
```

2. Write a C program that demonstrates the Multiple Threads with Global and Local Variables

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

#define NUM_THREADS 2

int global_variable = 0;

void *modifyGlobal(void *arg) {
    for (int i = 0; i < 5; i++) {
        global_variable++;
        printf("Thread %ld: Global Variable = %d\n", (long)arg, global_variable);
    }
    pthread_exit(NULL);
}

int main() {
    pthread_t threads[NUM_THREADS];

    for (long i = 0; i < NUM_THREADS; i++) {
        if (pthread_create(&threads[i], NULL, modifyGlobal, (void *)i) != 0) {
            perror("pthread_create");
            exit(EXIT_FAILURE);
        }
    }

    for (int i = 0; i < NUM_THREADS; i++) {
        pthread_join(threads[i], NULL);
    }

    int local_variable = 100;
    printf("Main Thread: Global Variable = %d, Local Variable = %d\n", global_variable,
local_variable);

    return 0;
}
```

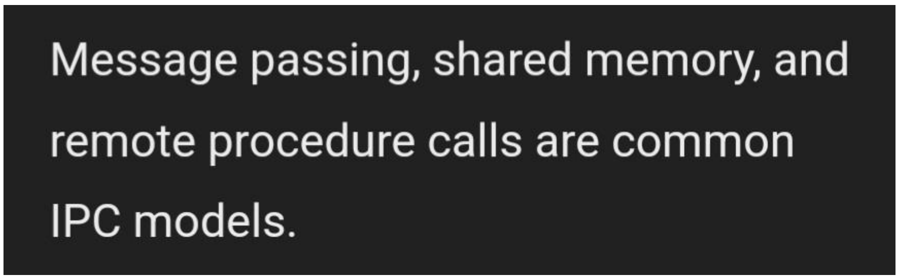
Sample VIVA-VOCE Questions (In-Lab):

1. Explain in detail about Inter-process Communication.



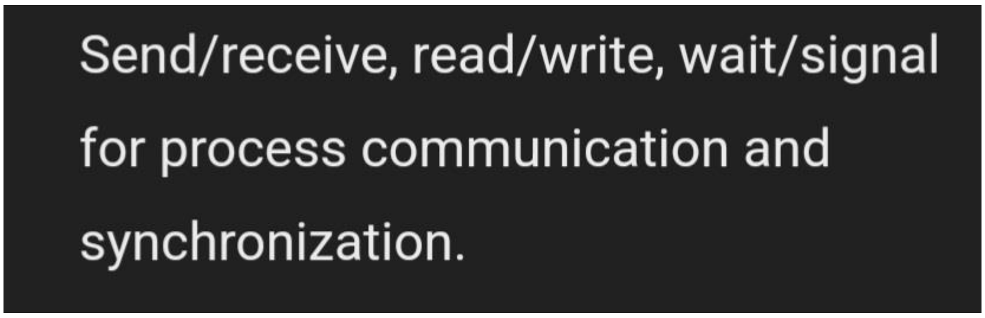
IPC allows processes to communicate using shared memory, message passing, and synchronization.

2. Explain in detail about models of IPC.



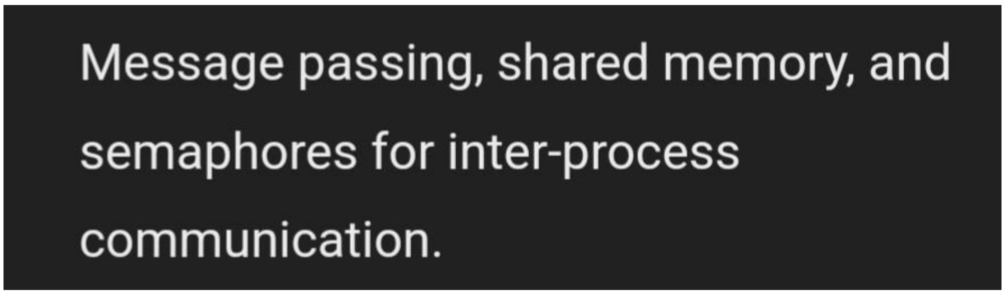
Message passing, shared memory, and remote procedure calls are common IPC models.

3. Write operations provided in IPC.



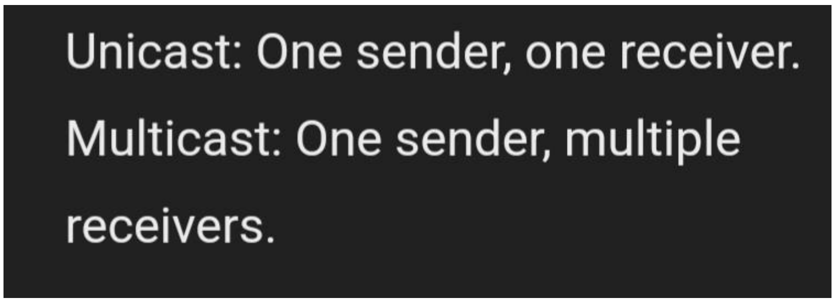
Send/receive, read/write, wait/signal for process communication and synchronization.

4. State IPC paradigms and implementations in OS.



Message passing, shared memory, and semaphores for inter-process communication.

5. What do you mean by "unicast" and "multicast" IPC?



Unicast: One sender, one receiver. Multicast: One sender, multiple receivers.

Evaluator Remark (if any):	**Marks Secured _____ out of 50**
	Signature of the Evaluator with Date

Note: Evaluator MUST ask Viva-voce before signing and posting marks for each experiment.