**COMSATS INSTITUTE OF INFORMATION TECHNOLOGY,**

**ISLAMABAD.**

Department of Computer Science

OPERATING SYSTEM CONCEPTS – CSC322

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Lab ASSIGNMENT 02**

Mapped to CLO2

MARKS: 30

**Instruction:**

* This is a Individual Assignment, Submit written report having codes and screen shots of output.
* You can consult Book for better understanding.
* Copying assignment will be graded zero.

1. Implement Merge sort in shared memory

#include <sys/types.h>

#include <sys/ipc.h>

#include <sys/shm.h>

#include <stdio.h>

#include <stdlib.h>

#include<sys/wait.h>

#include <unistd.h>

#include<ctime>

void insertionSort(int arr[], int n);

void merge(int a[], int l1, int h1, int h2);

void mergeSort(int a[], int l, int h)

{

int i, len=(h-l+1);

// Using insertion sort for small sized array

if (len<=5)

{

insertionSort(a+l, len);

return;

}

pid\_t lpid,rpid;

lpid = fork();

if (lpid<0)

{

// Lchild proc not created

perror("Left Child Proc. not created\n");

\_exit(-1);

}

else if (lpid==0)

{

mergeSort(a,l,l+len/2-1);

\_exit(0);

}

else

{

rpid = fork();

if (rpid<0)

{

// Rchild proc not created

perror("Right Child Proc. not created\n");

\_exit(-1);

}

else if(rpid==0)

{

mergeSort(a,l+len/2,h);

\_exit(0);

}

}

int status;

// Wait for child processes to finish

waitpid(lpid, &status, 0);

waitpid(rpid, &status, 0);

// Merge the sorted subarrays

merge(a, l, l+len/2-1, h);

}

/\* Function to sort an array using insertion sort\*/

void insertionSort(int arr[], int n)

{

int i, key, j;

for (i = 1; i < n; i++)

{

key = arr[i];

j = i-1;

/\* Move elements of arr[0..i-1], that are

greater than key, to one position ahead

of their current position \*/

while (j >= 0 && arr[j] > key)

{

arr[j+1] = arr[j];

j = j-1;

}

arr[j+1] = key;

}

}

// Method to merge sorted subarrays

void merge(int a[], int l1, int h1, int h2)

{

// We can directly copy the sorted elements

// in the final array, no need for a temporary

// sorted array.

int count=h2-l1+1;

int sorted[count];

int i=l1, k=h1+1, m=0;

while (i<=h1 && k<=h2)

{

if (a[i]<a[k])

sorted[m++]=a[i++];

else if (a[k]<a[i])

sorted[m++]=a[k++];

else if (a[i]==a[k])

{

sorted[m++]=a[i++];

sorted[m++]=a[k++];

}

}

while (i<=h1)

sorted[m++]=a[i++];

while (k<=h2)

sorted[m++]=a[k++];

int arr\_count = l1;

for (i=0; i<count; i++,l1++)

a[l1] = sorted[i];

}

// To check if array is actually sorted or not

void isSorted(int arr[], int len)

{

if (len==1)

{

printf("Sorting Done Successfully\n");

return;

}

int i;

for (i=1; i<len; i++)

{

if (arr[i]<arr[i-1])

{

printf("Sorting Not Done\n");

return;

}

}

printf("Sorting Done Successfully\n");

return;

}

// To fill randome values in array for testing

// purpise

void fillData(int a[], int len)

{

// Create random arrays

int i;

for (i=0; i<len; i++)

a[i] = rand();

return;

}

// Driver code

int main()

{

int shmid;

key\_t key = IPC\_PRIVATE;

int \*shm\_array;

// Using fixed size array. We can uncomment

// below lines to take size from user

int length = 128;

/\* printf("Enter No of elements of Array:");

scanf("%d",&length); \*/

// Calculate segment length

size\_t SHM\_SIZE = sizeof(int)\*length;

// Create the segment.

if ((shmid = shmget(key, SHM\_SIZE, IPC\_CREAT | 0666)) < 0)

{

perror("shmget");

\_exit(1);

}

// Now we attach the segment to our data space.

if ((shm\_array = (int \*)shmat(shmid, NULL, 0)) == (int \*) -1)

{

perror("shmat");

\_exit(1);

}

// Create a random array of given length

srand(time(NULL));

fillData(shm\_array, length);

// Sort the created array

mergeSort(shm\_array, 0, length-1);

// Check if array is sorted or not

isSorted(shm\_array, length);

/\* Detach from the shared memory now that we are

done using it. \*/

if (shmdt(shm\_array) == -1)

{

perror("shmdt");

\_exit(1);

}

/\* Delete the shared memory segment. \*/

if (shmctl(shmid, IPC\_RMID, NULL) == -1)

{

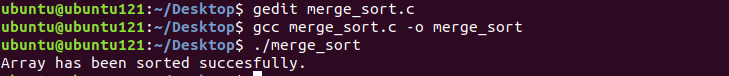
perror("shmctl");

\_exit(1);

}

return 0;

}



1. Implement chat application using message queue

**Sender**

#include <signal.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <sys/ipc.h>

#include <sys/shm.h>

#include <sys/types.h>

#include <unistd.h>

#define FILLED 0

#define Ready 1

#define NotReady -1

struct memory {

char buff[100];

int status, pid1, pid2;

};

struct memory\* shmptr;

// handler function to print message received from user2

void handler(int signum)

{

// if signum is SIGUSR1, then user 1 is receiving a message from user2

if (signum == SIGUSR1) {

printf("Received User2: ");

puts(shmptr->buff);

}

}

int main()

{

// process id of user1

int pid = getpid();

int shmid;

// key value of shared memory

int key = 12345;

// shared memory create

shmid = shmget(key, sizeof(struct memory), IPC\_CREAT | 0666);

// attaching the shared memory

shmptr = (struct memory\*)shmat(shmid, NULL, 0);

// store the process id of user1 in shared memory

shmptr->pid1 = pid;

shmptr->status = NotReady;

// calling the signal function using signal type SIGUSER1

signal(SIGUSR1, handler);

while (1) {

while (shmptr->status != Ready)

continue;

sleep(1);

// taking input from user1

printf("User1: ");

fgets(shmptr->buff, 100, stdin);

shmptr->status = FILLED;

// sending the message to user2 using kill function

kill(shmptr->pid2, SIGUSR2);

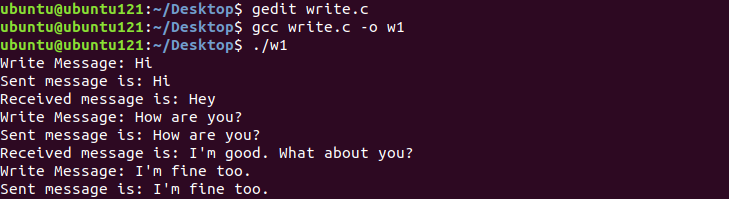
}

shmdt((void\*)shmptr);

shmctl(shmid, IPC\_RMID, NULL);

return 0;

}



**Receiver**

#include <signal.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <sys/ipc.h>

#include <sys/shm.h>

#include <sys/types.h>

#include <unistd.h>

#define FILLED 0

#define Ready 1

#define NotReady -1

struct memory {

char buff[100];

int status, pid1, pid2;

};

struct memory\* shmptr;

// handler function to print message received from user1

void handler(int signum)

{

// if signum is SIGUSR2, then user 2 is receiving a message from user1

if (signum == SIGUSR2) {

printf("Received From User1: ");

puts(shmptr->buff);

}

}

// main function

int main()

{

// process id of user2

int pid = getpid();

int shmid;

// key value of shared memory

int key = 12345;

// shared memory create

shmid = shmget(key, sizeof(struct memory), IPC\_CREAT | 0666);

// attaching the shared memory

shmptr = (struct memory\*)shmat(shmid, NULL, 0);

// store the process id of user2 in shared memory

shmptr->pid2 = pid;

shmptr->status = NotReady;

// calling the signal function using signal type SIGUSR2

signal(SIGUSR2, handler);

while (1) {

sleep(1);

// taking input from user2

printf("User2: ");

fgets(shmptr->buff, 100, stdin);

shmptr->status = Ready;

// sending the message to user1 using kill function

kill(shmptr->pid1, SIGUSR1);

while (shmptr->status == Ready)

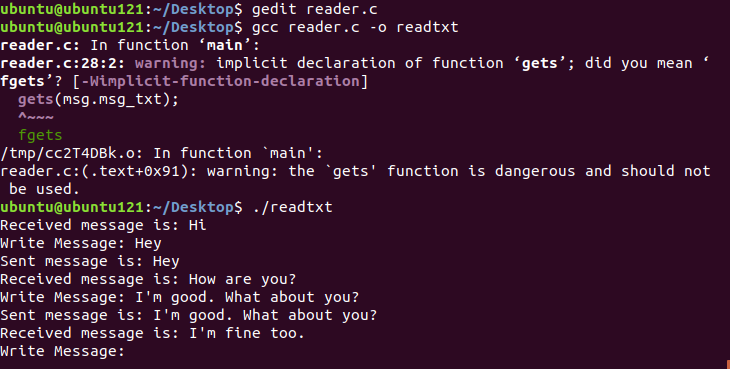
continue;

}

shmdt((void\*)shmptr);

return 0;

}



1. Write a program that uses multiple threads to find which integer between 1 and 100,000 has the largest number of divisors, and how many divisors does it have?. By using threads, your program will take less time to do the computation when it is run on a multiprocessor computer. At the end of the program, output the elapsed time, the integer that has the largest number of divisors, and the number of divisors that it has.

Q2. Can a process transition from waiting for an I/O operation to the terminated state?

Why or why not?

Q3. What are the differences between user-level and kernel-level threads? Under what

circumstances is one type better than the other? What is the essential cause of the

difference in cost between a context switch for kernel-level threads and a switch that

occurs between user-level threads?

Q2. Can a process transition from waiting for an I/O operation to the terminated state?

Why or why not?

Q3. What are the differences between user-level and kernel-level threads? Under what

circumstances is one type better than the other? What is the essential cause of the

difference in cost between a context switch for kernel-level threads and a switch that

occurs between user-level threads?