Copy code

// Close file

fclose(fp);

Command Line Arguments

c

Copy code

// Declare main function with arguments

int main(int argc, char \*argv[]) {

// Your code here

return 0;

}

// Access command line arguments

printf("%s", argv[1]); // Output: first argument

POSIX Thread Implementation

Functions of pthread

c

Copy code

// Create thread

pthread\_t thread\_id;

pthread\_create(&thread\_id, NULL, function\_name, NULL);

// Join thread

pthread\_join(thread\_id, NULL);

// Exit thread

pthread\_exit(NULL);

Sure, here is a cheat sheet covering the topics you mentioned:

How to Write a C Program

c

Copy code

// Include standard input/output header file

#include <stdio.h>

// Define main function

int main() {

// Your code here

return 0;

}

Data Types in C

Data Type Size (in bytes) Description

char 1 Character or small integer

int 2 or 4 Integer

float 4 Floating-point number

double 8 Double precision floating-point number

Pointer in C

c

Copy code

// Declare pointer variable

int \*p;

// Assign value to pointer

int x = 5;

p = &x;

// Dereference pointer to get value

printf("%d", \*p); // Output: 5

Array in C

c

Copy code

// Declare array

int arr[5];

// Assign values to array

arr[0] = 1;

arr[1] = 2;

arr[2] = 3;

arr[3] = 4;

arr[4] = 5;

// Access array elements

printf("%d", arr[2]); // Output: 3

String in C

c

Copy code

// Declare string

char str[6] = "Hello";

// Print string

printf("%s", str); // Output: Hello

Flow Statement in C

Loops in C

c

Copy code

// For loop

for (int i = 0; i < 5; i++) {

// Your code here

}

// While loop

int i = 0;

while (i < 5) {

// Your code here

i++;

}

// Do-while loop

int i = 0;

do {

// Your code here

i++;

} while (i < 5);

Function in C

c

Copy code

// Declare function

int add(int a, int b) {

return a + b;

}

// Call function

int sum = add(2, 3);

printf("%d", sum); // Output: 5

FILE I/O

Open

c

Copy code

// Open file for reading

FILE \*fp = fopen("file.txt", "r");

// Open file for writing

FILE \*fp = fopen("file.txt", "w");

Read

c

Copy code

// Read character from file

int ch = fgetc(fp);

// Read string from file

char str[100];

fgets(str, 100, fp);

Write

c

Copy code

// Write character to file

fputc('a', fp);

// Write string to file

char str[6] = "Hello";

fputs(str, fp);

Close

c

Copy code

// Close file

fclose(fp);

Command Line Arguments

c

Copy code

// Declare main function with arguments

int main(int argc, char \*argv[]) {

// Your code here

return 0;

}

// Access command line arguments

printf("%s", argv[1]); // Output: first argument

POSIX Thread Implementation

Functions of pthread

c

Copy code

// Create thread

pthread\_t thread\_id;

pthread\_create(&thread\_id, NULL, function\_name, NULL);

// Join thread

pthread\_join(thread\_id, NULL);

// Exit thread

pthread\_exit(NULL);

**//Assignment-02: Threading Tasks**

**//Task01**

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

#include <pthread.h> #include <unistd.h>

Write a c program that creates 5 threads and prints which thread is running and after the thread is closed, a new thread starts its execution. Each thread should run sequentially one by one.

#define NUM\_THREADS 5

// Thread function

void\* thread\_func(void\* arg) {

int thread\_num = \*((int\*) arg);

printf("thread-%d running\n", thread\_num);

printf("thread-%d closed\n", thread\_num);

return NULL;

}

// Main function

int main() {

pthread\_t threads[NUM\_THREADS];

int thread\_nums[NUM\_THREADS];

for (int i = 0; i < NUM\_THREADS; i++) {

thread\_nums[i] = i + 1;

pthread\_create(&threads[i], NULL, thread\_func, (void\*) &thread\_nums[i]);

pthread\_join(threads[i], NULL);

}

return 0;

}

**//Task02**

Write a program in c using 5 threads where each thread will print 5 integers

The outputs will look like this:

Thread 1: 1 2 3 4 5

Thread 2: 6 7 8 9 10

Thread 3: 11 12 13 14 15

Thread 4: 16 17 18 19 20

Thread 5: 21 22 23 24 25

void \*thread\_function(void \*arg)

{

int \*id = (int \*)arg;

for (int i = 0; i < 5; i++)

{

if (\*id == 1) {

printf("Thread %d prints %d\n", \*id, i + 1);

} else if (\*id == 2 {

printf("Thread %d prints %d\n", \*id, i + 6);

} else if (\*id == 3) {

printf("Thread %d prints %d\n", \*id, i + 11);

} else if (\*id == 4) {

printf("Thread %d prints %d\n", \*id, i + 16);

} else if (\*id == 5) {

printf("Thread %d prints %d\n", \*id, i + 21);

}

}

pthread\_exit(NULL);

}

int main()

{

pthread\_t thread\_id[5];

int i;

int \*id;

for (i = 0; i < 5; i++) {

id = (int \*)malloc(sizeof(int));

\*id = i + 1;

pthread\_create(&thread\_id[i], NULL, thread\_function, id);

pthread\_join(thread\_id[i], NULL);

}

return 0;

}

**//Task03**

Write a program in c that has a function that takes the name of the user and adds all the ASCII value of the characters and returns it. Now create 3 threads that run the function using 3 different user names. Now print “Youreka” if all the returned values are equal, print “Miracle” if any 2 returned values are equal and print “Hasta la vista” if the values don’t match using another thread.

// function to add all the ASCII values of the characters

void \*count\_ascii(void \*arg)

{

char \*name = (char \*)arg;

int i = 0, sum = 0;

int \*result = (int \*)malloc(sizeof(int));

while (name[i] != '\0') {

sum += name[i];

// printf("%c", name[i]);

// printf("%d", sum);

i++;

}

// prinnf("%s: %d\n", name, sum);

\*result = sum;

return result;

}

int main()

{

pthread\_t t1, t2, t3;

char \*name1 = "Iftekhar";

char \*name2 = "Hossain";

char \*name3 = "Turja";

int \*result1, \*result2, \*result3;

// create 3 threads to run the function

pthread\_create(&t1, NULL, count\_ascii, (void \*)name1);

pthread\_join(t1, (void \*\*)&result1);

pthread\_create(&t2, NULL, count\_ascii, (void \*)name2);

pthread\_join(t2, (void \*\*)&result2);

pthread\_create(&t3, NULL, count\_ascii, (void \*)name3);

pthread\_join(t3, (void \*\*)&result3);

// compare the results and print the appropriate message

if (\*result1 == \*result2 && \*result1 == \*result3) {

printf("Youreka\n");

} else if (\*result1 == \*result2 || \*result1 == \*result3 || \*result2 == \*result3) {

printf("Miracle\n");

} else {

printf("Hasta la vista\n");

}

return 0;

}

printf("Odd/Even status of the numbers: ");

for(int i=0; i<n; i++) {

int num = arr[i];

if(num % 2 == 0) {

printf("Even ");

} else {

printf("Odd ");

}

}

printf("\n");

}

return 0;

}

/\* Bash sample commands:

# Compile sort.c

gcc -o sort sort.c

# Run sort.c with some numbers as command line arguments

./sort 5 2 8 1 6

# Compile oddeven.c

gcc -o oddeven oddeven.c

# Run oddeven.c with some numbers as command line arguments

./oddeven 5 2 8 1 6

# Compile parent\_child.c

gcc -o parent\_child parent\_child.c

# Run parent\_child.c with some numbers as command line arguments

./parent\_child 5 2 8 1 6

\*/

**//Task05**

#include <stdio.h> #include <unistd.h> #include <sys/types.h>

Write a program in c that the parent process will create one child process and 3 grandchild

processes and print their IDs

Output: 1. Parent process ID : 0

2. Child process ID : ...

3. Grand Child process ID: ...

4. Grand Child process ID: ...

5. Grand Child process ID: ...

int main() {

pid\_t pid, child\_pid, grandchild\_pid1, grandchild\_pid2, grandchild\_pid3;

pid = getpid(); // Get parent process ID

printf("1. Parent process ID: %d\n", pid);

// Create child process

child\_pid = fork();

if (child\_pid == 0) { // Child process

pid = getpid(); // Get child process ID

printf("2. Child process ID: %d\n", pid);

// Create three grandchild processes

grandchild\_pid1 = fork();

if (grandchild\_pid1 == 0) { // Grandchild process 1

pid = getpid(); // Get grandchild process ID

printf("3. Grandchild process ID: %d\n", pid);

return 0;

}

grandchild\_pid2 = fork();

if (grandchild\_pid2 == 0) { // Grandchild process 2

pid = getpid(); // Get grandchild process ID

printf("4. Grandchild process ID: %d\n", pid);

return 0;

}

grandchild\_pid3 = fork();

if (grandchild\_pid3 == 0) { // Grandchild process 3

pid = getpid(); // Get grandchild process ID

printf("5. Grandchild process ID: %d\n", pid);

return 0;

}

return 0;

}

return 0;

}

**Lecture 5**

**POSIX Thread**:

In Linux platforms, the C language contains pthread standard API for all kinds of thread related functions.

It is also known as a POSIX thread that allows users to create many threads for simultaneous processes to

flow. It is best used in multi-core systems or processors to implement threads on the kernel to achieve

the system.

**Implementation**:

(Library: #include <pthread.h>)

It is necessary to include this pthread.h header file in the script initially. This will help in using the functions

of the pthread library. To compile and execute the c file consisting of pthread library following commands

should be used. (Assume the name of the c file is th.c)

gcc -o th th.c -lpthread

./th

**Functions of pthread:**

1. int pthread\_create(pthread\_t \* thread\_id, const pthread\_attributes\_t \* attr, void \*

(\*thread\_function), void \*argument);

Purpose: It is used to create a new thread.

Parameters:

▪ thread\_id: This acts as a pointer to an unsigned integer value. It returns the thread id of the

thread that is formed.

▪ attributes: This parameter acts as a pointer to a structure. It is used to define attributes of a

thread that can be the policy of scheduling, and stack address, etc.

▪ thread\_function: This parameter is a pointer to the thread function of a thread.

▪ argument: This parameter is a pointer to void with different arguments to the function predefined at the start of the argument.

2. void pthread\_exit(void \*return\_value);

Purpose: It is used to terminate or end a thread.

Parameters:

▪ return\_value: It stores the status of the thread such that the thread terminates. It must be a

global variable. This will allow the next thread in line to join the thread if it is available.

3. int pthread\_join(pthread\_t thread\_id, void \*thread\_return);

Purpose: This is a function used at the time of wait for the termination of the thread.

Parameters:

▪ thread\_id: It is the ID of the thread for which the thread in line waits.

▪ thread\_return: It is the parameter tha

while (strcmp(str, "-1") != 0) {

// Write string to file

fprintf(fp, "%s\n", str);

// Ask user for another string

printf("Enter another string (enter '-1' to quit):\n");

scanf("%s", str);

}

// Close file

fclose(fp);

printf("File %s has been written to.\n", filename);

return 0;

}

/\* Bash sample commands:

# Compile task01.c

gcc -o t01 task01.c

# Run sort.c with filename as command line arguments

./t01 filename

\*/

**//Task02**

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

Write a program that will create a child and another grandchild process. Every process will print

a line.

Parent process will print, “I am parent”

Child process will print, “I am child”

Grandchild process will print, “I am grandchild”

Now, write the program in such way so that the following output will be shown -

I am grandchild

I am child

I am parent

int main() {

// create child process

pid\_t child\_pid = fork();

if (child\_pid == 0) {

// child process

pid\_t grandchild\_pid = fork();

if (grandchild\_pid == 0) {

// grandchild process

printf("I am grandchild\n");

} else {

// child process

wait(NULL);

printf("I am child\n");

}

} else {

// parent process

wait(NULL);

printf("I am parent\n");

}

return 0;

}

**//Task03**

#include <stdio.h> #include <unistd.h> #include <sys/wait.h>

Consider the following code snippet in your main function -

a = fork(); b = fork(); c = fork();

Now, write the full program, that will check the children’s PID for odd or even and if it is odd then the process will create another child process. Lastly, print how many processes have been created considering the first parent process.

int main() {

int a, b, c;

int process\_count = 1; // initialize process count to 1 for the first parent process

a = fork(); b = fork(); c = fork();

if (a == 0) {

// child process a

if (getpid() % 2 != 0) {

// PID is odd, create another child process

fork();

process\_count++;

}

} else if (b == 0) {

// child process b

if (getpid() % 2 != 0) {

// PID is odd, create another child process

fork();

process\_count++;

}

} else if (c == 0) {

// child process c

if (getpid() % 2 != 0) {

// PID is odd, create another child process

fork();

process\_count++;

}

} else {

// parent process

process\_count += 3; // add 3 for the child processes created by fork()

printf("Total number of processes created: %d\n", process\_count);

}

return 0;

}

**//Task04**

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

Write a program that will create a child process and the child process will first sort the array that you have declared in this program. And then, the parent process will print the odd/even status for each number in the array.

int compare(const void \* a, const void \* b) {

return (\*(int\*)b - \*(int\*)a);

}

int main(int argc, char \*argv[]) {

int arr[argc-1];

for(int i=0; i<argc-1; i++) {

arr[i] = atoi(argv[i+1]);

}

int n = argc-1;

pid\_t pid = fork();

if(pid == 0) {

// Child process

qsort(arr, n, sizeof(int), compare);

printf("Sorted array in descending order: ");

for(int i=0; i<n; i++) {

printf("%d ", arr[i]);

}

printf("\n");

} else {

// Parent process

wait(NULL);

printf("Odd/Even status of the numbers: ");

for(int i=0; i<n; i++) {

int num = arr[i];

if(num % 2 == 0) {

printf("Even ");

} else {

printf("Odd ");

}

}

printf("\n");

}

return 0;

}

/\* Bash sample commands:

# Compile sort.c

gcc -o sort sort.c

# Run sort.c with some numbers as command line arguments

./sort 5 2 8 1 6

# Compile oddeven.c

gcc -o oddeven oddeven.c

# Run oddeven.c with some numbers as command line arguments

./oddeven 5 2 8 1 6

# Compile parent\_child.c

gcc -o parent\_child parent\_child.c

# Run parent\_child.c with some numbers as command line arguments

./parent\_child 5 2 8 1 6

\*/

//Task05

#include <stdio.h> #include <unistd.h> #include <sys/types.h>

Write a program in c that the parent process will create one child process and 3 grandchild

processes and print their IDs

Output: 1. Parent process ID : 0

2. Child process ID : ...

3. Grand Child process ID: ...

4. Grand Child process ID: ...

5. Grand Child process ID: ...

int main() {

pid\_t pid, child\_pid, grandchild\_pid1, grandchild\_pid2, grandchild\_pid3;

pid = getpid(); // Get parent process ID

printf("1. Parent process ID: %d\n", pid);

// Create child process

child\_pid = fork();

if (child\_pid == 0) { // Child process

pid = getpid(); // Get child process ID

printf("2. Child process ID: %d\n", pid);

// Create three grandchild processes

grandchild\_pid1 = fork();

if (grandchild\_pid1 == 0) { // Grandchild process 1

pid = getpid(); // Get grandchild process ID

printf("3. Grandchild process ID: %d\n", pid);

return 0;

}

grandchild\_pid2 = fork();

if (grandchild\_pid2 == 0) { // Grandchild process 2

pid = getpid(); // Get grandchild process ID

printf("4. Grandchild process ID: %d\n", pid);

return 0;

}

grandchild\_pid3 = fork();

if (grandchild\_pid3 == 0) { // Grandchild process 3

pid = getpid(); // Get grandchild process ID

printf("5. Grandchild process ID: %d\n", pid);

return 0;

}

return 0;

}

return 0;

}

**// Assignment-02: (struct and function)**

**//Task01**

#include <stdio.h>

Suppose you and your friends go to have breakfast in a restaurant. Each of you order Paratha, Vegetable and Mineral Water. Treat each of the ordered items as structures and each of the structures will have two properties which are: quantity and unit price. Each property of the structure will be taken as input from the user. After taking all the inputs calculate what is the total bill and also, take input from the user how many people are there in total. Lastly calculate how much each of the person will have to pay and print it (Note: This value will be float).

// Structure of items

struct item {

int quantity;

float unit\_price;

};

int main() {

struct item paratha, vegetable, water;

int num\_people;

float total\_bill, bill\_per\_person;

// Take input for Paratha

printf("Quantity Of Paratha: ");

scanf("%d", &paratha.quantity);

printf("Unit Price: ");

scanf("%f", &paratha.unit\_price);

// Take input for Vegetable

printf("Quantity Of Vegetables: ");

scanf("%d", &vegetable.quantity);

printf("Unit Price: ");

scanf("%f", &vegetable.unit\_price);

// Take input for Mineral Water

printf("Quantity Of Mineral Water: ");

scanf("%d", &water.quantity);

printf("Unit Price: ");

scanf("%f", &water.unit\_price);

// Calculate total bill

total\_bill = (paratha.quantity \* paratha.unit\_price) + (vegetable.quantity \* vegetable.unit\_price) + (water.quantity \* water.unit\_price);

// Take input for number of people

printf("Number of People: ");

scanf("%d", &num\_people);

// Calculate bill per person

bill\_per\_person = total\_bill / num\_people;

// Print result

printf("Individual people will pay: %.2f tk\n", bill\_per\_person);

return 0;

}

**//Task02**

#include <stdio.h>

Write a C program to print perfect numbers between given intervals using a function. A perfect number is a positive integer equal to the sum of its positive divisors, excluding the number itself.

// Function of perfect numbers

int is\_perfect(int n) {

int i, sum = 0;

// Find divisors and add them up

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

if (n % i == 0) {

sum += i;

}

}

// Check if sum of divisors equals the number itself

if (sum == n) {

return 1;

} else {

return 0;

}

}

int main() {

int start, end, i;

// Take input for range

printf("Enter start of range: ");

scanf("%d", &start);

printf("Enter end of range: ");

scanf("%d", &end);

// Print perfect numbers in range

printf("Perfect numbers in range %d to %d:\n", start, end);

for (i = start; i <= end; i++) {

if (is\_perfect(i)) {

printf("%d\n", i);

}

}

return 0;

}

**//Assignment-02: System Call Tasks**

**//Task01**

#include <stdio.h> #include <stdlib.h> #include <string.h>

Write a c program that will open a file given from the command line argument and then it will ask the user to input strings which will be written to that file. It will continue to ask the user to enter a string as long as the user enters “-1”. If the given file does not exist in the directory, then your program will automatically create the file. All code should be Unix compatible.

// main function

int main(int argc, char \*argv[]) {

FILE \*fp;

char filename[100], str[100];

// Check if filename was provided as command line argument

if (argc < 2) {

printf("Please provide a filename as command line argument.\n");

exit(1);

}

// Get filename from command line argument

strcpy(filename, argv[1]);

// Open file in write mode

fp = fopen(filename, "w");

// Check if file was opened successfully

if (fp == NULL) {

printf("Unable to open file %s.\n", filename);

exit(1);

}

// Ask user to input strings

printf("Enter a string to write to file (enter '-1' to quit):\n");

scanf("%s", str);

while (strcmp(str, "-1") != 0) {

// Write string to file

fprintf(fp, "%s\n", str);

// Ask user for another string

printf("Enter another string (enter '-1' to quit):\n");

scanf("%s", str);

}

// Close file

fclose(fp);

printf("File %s has been written to.\n", filename);

return 0;

}

/\* Bash sample commands:

# Compile task01.c

gcc -o t01 task01.c

# Run sort.c with filename as command line arguments

./t01 filename

\*/

//Task02

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

Write a program that will create a child and another grandchild process. Every process will print

a line.

Parent process will print, “I am parent”

Child process will print, “I am child”

Grandchild process will print, “I am grandchild”

Now, write the program in such way so that the following output will be shown -

I am grandchild

I am child

I am parent

int main() {

// create child process

pid\_t child\_pid = fork();

if (child\_pid == 0) {

// child process

pid\_t grandchild\_pid = fork();

if (grandchild\_pid == 0) {

// grandchild process

printf("I am grandchild\n");

} else {

// child process

wait(NULL);

printf("I am child\n");

}

} else {

// parent process

wait(NULL);

printf("I am parent\n");

}

return 0;

}