Q.1 File Management using ‘open’, ‘read’, and ‘write’ system calls

Source Code:

#include <stdio.h>

#include <stdlib.h>

#include <fcntl.h>

#include <unistd.h>

int main() {

int fd;

char buffer[100];

fd = open("sample.txt", O\_CREAT | O\_WRONLY, S\_IRUSR | S\_IWUSR);

if (fd == -1) {

perror("open");

exit(1);

}

write(fd, "satyam, GEHU", 13);

close(fd);

fd = open("sample.txt", O\_RDONLY);

if (fd == -1) {

perror("open");

exit(1);

}

read(fd, buffer, sizeof(buffer));

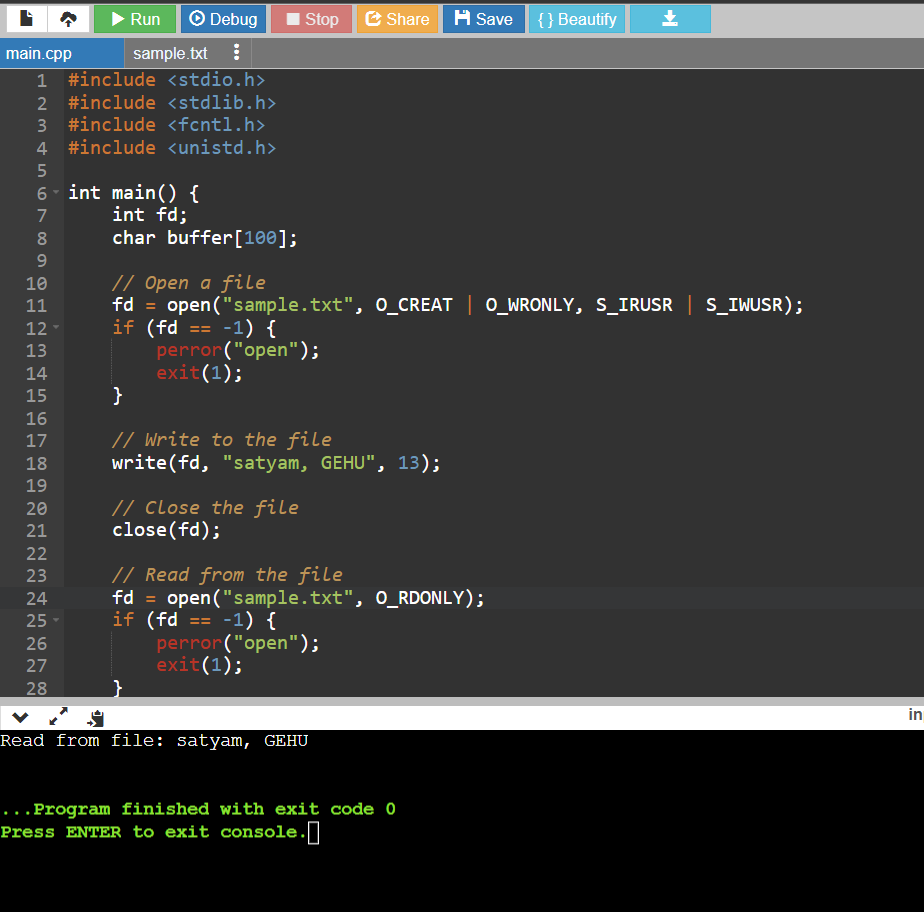
printf("Read from file: %s\n", buffer);

close(fd);

return 0;

}

Output:





Q2 Implementation of Process Control system calls

Source Code:

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/wait.h>

void create\_process() {

pid\_t pid = fork();

if (pid == -1) {

perror("Fork failed");

exit(EXIT\_FAILURE);

} else if (pid == 0) {

printf("Child process: PID=%d\n", getpid());

exit(EXIT\_SUCCESS);

} else {

printf("Parent process: Child PID=%d\n", pid);

int status;

waitpid(pid, &status, 0);

if (WIFEXITED(status)) {

printf("Child process terminated with status: %d\n", WEXITSTATUS(status));

}

}

}

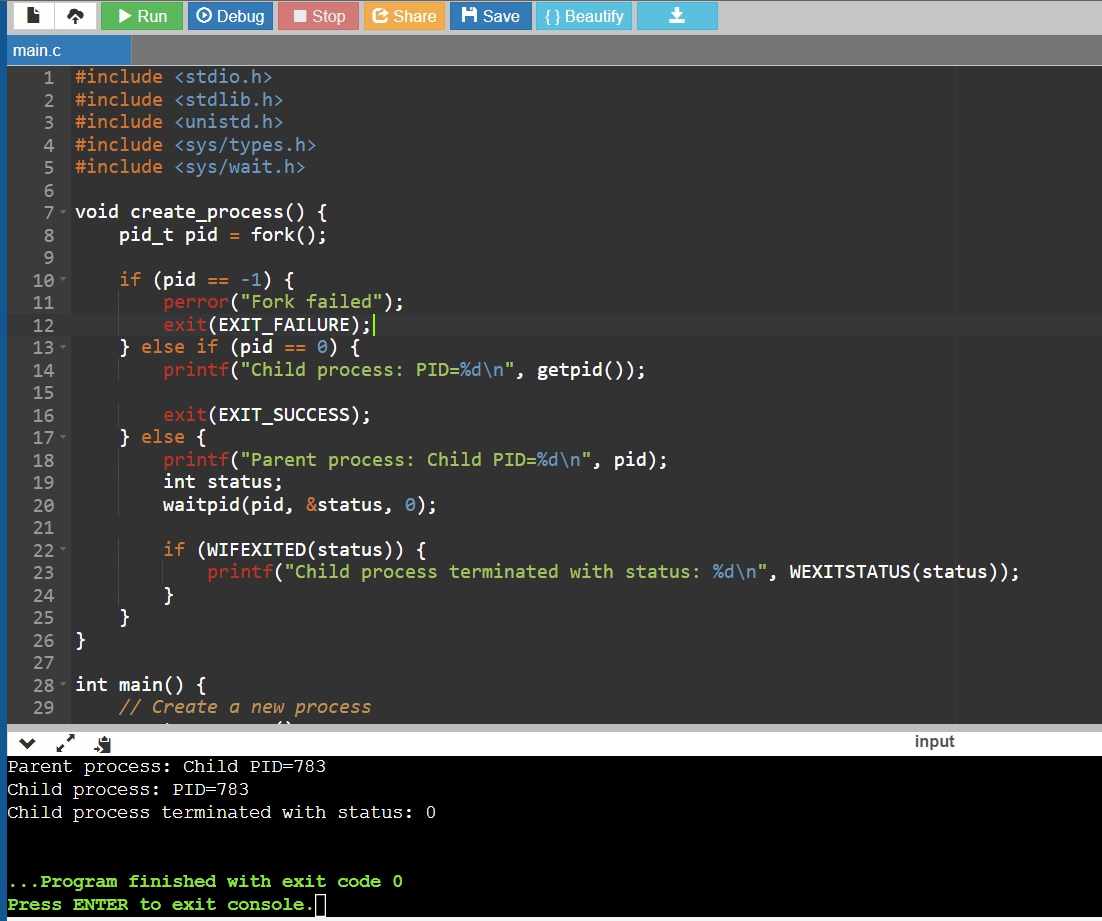
int main() {

// Create a new process

create\_process();

return 0;

Output:



Q3. Implementation of Device management system call

Source Code:

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

typedef enum {

PRINTER,

DISK\_DRIVE

} DeviceType;

typedef struct {

DeviceType type;

int id;

char\* name;

} Device;

int open\_device(DeviceType type, int id) {

printf("Device %d (Type: %d) opened.\n", id, type);

return 0;

}

int close\_device(int deviceID) {

printf("Device %d closed.\n", deviceID);

return 0;

}

int main() {

int printerID = open\_device(PRINTER, 1);

int diskDriveID = open\_device(DISK\_DRIVE, 2);

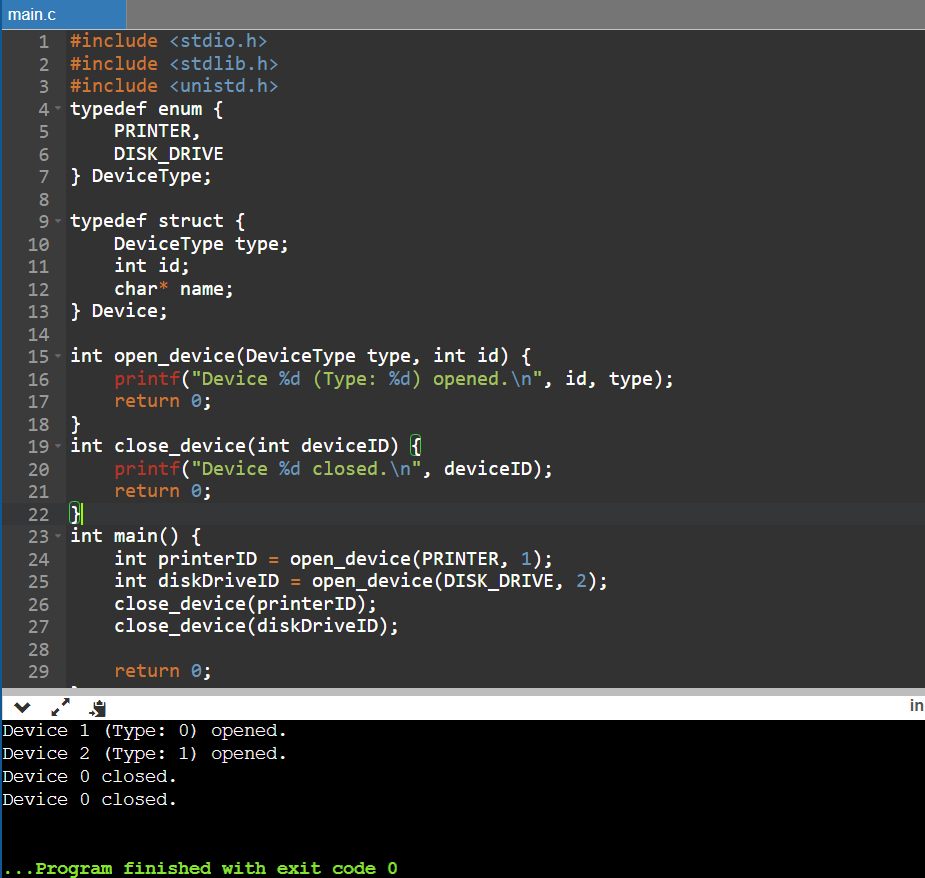
close\_device(printerID);

close\_device(diskDriveID);

return 0;

}

Output:



Q4. Implementation of Information maintenance system call

Source code:

#include <stdio.h>

#include <stdlib.h>

#include <sys/sysinfo.h>

typedef struct {

long total\_ram; // Total RAM in kilobytes

long free\_ram; // Free RAM in kilobytes

long total\_swap; // Total swap space in kilobytes

long free\_swap; // Free swap space in kilobytes

} SystemInfo;

int get\_system\_info(SystemInfo \*info) {

struct sysinfo si;

if (sysinfo(&si) != 0) {

perror("Failed to retrieve system information");

return -1;

}

info->total\_ram = si.totalram;

info->free\_ram = si.freeram;

info->total\_swap = si.totalswap;

info->free\_swap = si.freeswap;

return 0;

}

int main() {

SystemInfo info;

if (get\_system\_info(&info) == 0) {

printf("System Information:\n");

printf("Total RAM: %ld KB\n", info.total\_ram);

printf("Free RAM: %ld KB\n", info.free\_ram);

printf("Total Swap: %ld KB\n", info.total\_swap);

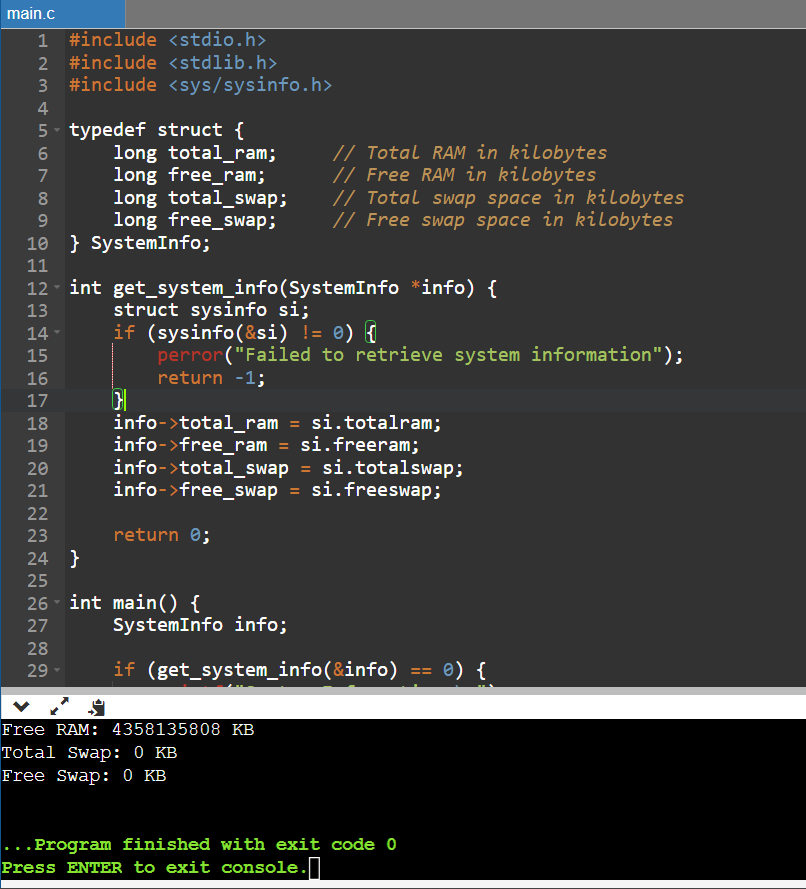
printf("Free Swap: %ld KB\n", info.free\_swap);

}

return 0;

}

Output:



Q5. Implementation of communication system call

Source Code:

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <string.h>

#define FIFO\_FILE "my\_fifo"

void writer\_process() {

int fd;

char message[] = "Hello I am Satyam";

// Create the FIFO (named pipe) if it doesn't exist

mkfifo(FIFO\_FILE, 0666);

fd = open(FIFO\_FILE, O\_WRONLY);

write(fd, message, strlen(message) + 1);

close(fd);

}

void reader\_process() {

int fd;

char message[100];

fd = open(FIFO\_FILE, O\_RDONLY);

read(fd, message, sizeof(message));

printf("Received message: %s\n", message);

close(fd);

}

int main() {

pid\_t pid;

// Create a child process

pid = fork();

if (pid == -1) {

perror("Fork failed");

exit(EXIT\_FAILURE);

} else if (pid == 0) {

// Child process (reader)

reader\_process();

} else {

// Parent process (writer)

writer\_process();

}

return 0;

}

Output:

