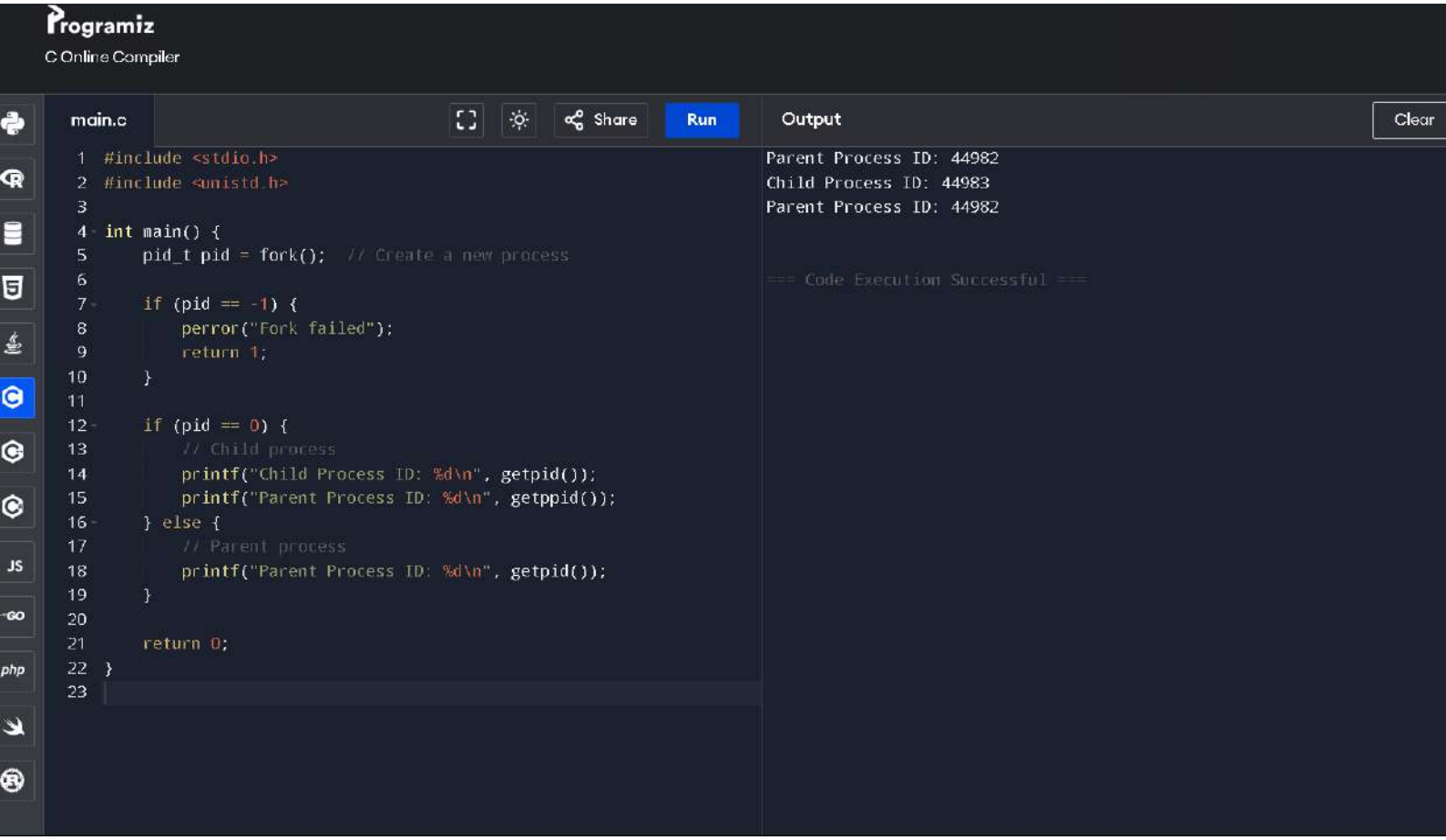
Programiz
C Online Compiler
main.c
Share
Run
Output
Clear
#include <stdio.h>
#include <unistd.h>
int main() {
pid_t pid = fork(); // Create a new process
if (pid == -1) {
perror("Fork failed");
return 1;
}
if (pid == 0) {
// Child process
printf("Child Process ID: %d\n", getpid());
printf("Parent Process ID: %d\n", getppid());
} else {
// Parent process
printf("Parent Process ID: %d\n", getpid());
}
return 0;
}
Parent Process ID: 44982
Child Process ID: 44983
Parent Process ID: 44982
=== Code Execution Successful ===

main.c

Share Run

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

void copyFile(const char *source, const char *destination) {
    FILE *src = fopen(source, "r");
    FILE *dest = fopen(destination, "w");

    if (src == NULL || dest == NULL) {
        perror("File opening failed");
        exit(1);
    }

    char ch;
    while ((ch = fgetc(src)) != EOF) {
        fputc(ch, dest);
    }

    fclose(src);
    fclose(dest);
}

int main() {
    copyFile("source.txt", "destination.txt");
    printf("File copied successfully.\n");
    return 0;
}
```

Output Clear

```
File opening failed: Permission denied


=== Code Exited With Errors ===
```

main.c

Share Run

```c
#include <stdio.h>

void FCFS(int burstTime[], int n) {
    int waitingTime[n], turnaroundTime[n];
    int totalWT = 0, totalTAT = 0;

    waitingTime[0] = 0;
    for (int i = 1; i < n; i++) {
        waitingTime[i] = burstTime[i - 1] + waitingTime[i - 1];
    }

    for (int i = 0; i < n; i++) {
        turnaroundTime[i] = burstTime[i] + waitingTime[i];
        totalWT += waitingTime[i];
        totalTAT += turnaroundTime[i];
    }

    printf("Process\tBurst Time\tWaiting Time\tTurnaround Time\n"
        );
    for (int i = 0; i < n; i++) {
        printf("%d\t%d\t\t%d\t\t%d\n", i+1, burstTime[i],
            waitingTime[i], turnaroundTime[i]);
    }

    printf("\nAverage Waiting Time: %.2f\n", (float)totalWT/n);
    printf("Average Turnaround Time: %.2f\n", (float)totalTAT/n);
}

```

Output

Clear

```
Process Burst Time  Waiting Time    Turnaround Time
1   6       0       6
2   8       6       14
3   7       14      21

Average Waiting Time: 6.67
Average Turnaround Time: 13.67


=== Code Execution Successful ===
```

main.c

Share Run

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

void SJF(int burstTime[], int n) {
    int waitingTime[n], turnaroundTime[n];
    int totalWT = 0, totalTAT = 0;

    // Sorting the burst time in increasing order
    for (int i = 0; i < n-1; i++) {
        for (int j = i+1; j < n; j++) {
            if (burstTime[i] > burstTime[j]) {
                int temp = burstTime[i];
                burstTime[i] = burstTime[j];
                burstTime[j] = temp;
            }
        }
    }

    waitingTime[0] = 0;
    for (int i = 1; i < n; i++) {
        waitingTime[i] = burstTime[i - 1] + waitingTime[i - 1];
    }

    for (int i = 0; i < n; i++) {
        turnaroundTime[i] = burstTime[i] + waitingTime[i];
        totalWT += waitingTime[i];
        totalTAT += turnaroundTime[i];
    }
```

Output Clear

```
Process Burst Time  Waiting Time    Turnaround Time
1   6       0       6
2   7       6       13
3   8       13      21

Average Waiting Time: 6.33
Average Turnaround Time: 13.33


=== Code Execution Successful ===
```

main.c

Share Run

```c
#include <stdio.h>

void priorityScheduling(int burstTime[], int priority[], int n) {
    int waitingTime[n], turnaroundTime[n];
    int totalWT = 0, totalTAT = 0;

    // Sorting processes based on priority
    for (int i = 0; i < n-1; i++) {
        for (int j = i+1; j < n; j++) {
            if (priority[i] > priority[j]) {
                int temp = priority[i];
                priority[i] = priority[j];
                priority[j] = temp;

                temp = burstTime[i];
                burstTime[i] = burstTime[j];
                burstTime[j] = temp;
            }
        }
    }

    waitingTime[0] = 0;
    for (int i = 1; i < n; i++) {
        waitingTime[i] = burstTime[i - 1] + waitingTime[i - 1];
    }

    for (int i = 0; i < n; i++) {
        turnaroundTime[i] = burstTime[i] + waitingTime[i];
```

Output Clear

```
Process Burst Time  Priority    Waiting Time    Turnaround Time
1   8       1       0       8
2   6       2       8       14
3   7       3       14      21

Average Waiting Time: 7.33
Average Turnaround Time: 14.33


=== Code Execution Successful ===
```

main.c

Share Run

```c
#include <stdio.h>

void preemptivePriorityScheduling(int burstTime[], int priority[]
    , int n) {
    int waitingTime[n], turnaroundTime[n];
    int totalWT = 0, totalTAT = 0;

    // Preemptive priority scheduling logic
    // Sorting processes based on priority
    for (int i = 0; i < n-1; i++) {
        for (int j = i+1; j < n; j++) {
            if (priority[i] > priority[j]) {
                int temp = priority[i];
                priority[i] = priority[j];
                priority[j] = temp;

                temp = burstTime[i];
                burstTime[i] = burstTime[j];
                burstTime[j] = temp;
            }
        }
    }

    waitingTime[0] = 0;
    for (int i = 1; i < n; i++) {
        waitingTime[i] = burstTime[i - 1] + waitingTime[i - 1];
    }
```

Output Clear

```
Process Burst Time  Priority    Waiting Time    Turnaround Time
1   8       1       0       8
2   6       2       8       14
3   7       3       14      21

Average Waiting Time: 7.33
Average Turnaround Time: 14.33


=== Code Execution Successful ===
```

main.c

Share Run

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

void SJF(int burstTime[], int n) {
    int waitingTime[n], turnaroundTime[n];
    int totalWT = 0, totalTAT = 0;

    // Sorting the burst time in increasing order
    for (int i = 0; i < n-1; i++) {
        for (int j = i+1; j < n; j++) {
            if (burstTime[i] > burstTime[j]) {
                int temp = burstTime[i];
                burstTime[i] = burstTime[j];
                burstTime[j] = temp;
            }
        }
    }

    waitingTime[0] = 0;
    for (int i = 1; i < n; i++) {
        waitingTime[i] = burstTime[i - 1] + waitingTime[i - 1];
    }

    for (int i = 0; i < n; i++) {
        turnaroundTime[i] = burstTime[i] + waitingTime[i];
        totalWT += waitingTime[i];
        totalTAT += turnaroundTime[i];
    }
```

Output Clear

```
Process Burst Time  Waiting Time    Turnaround Time
1   6       0       6
2   7       6       13
3   8       13      21

Average Waiting Time: 6.33
Average Turnaround Time: 13.33


=== Code Execution Successful ===
```

main.c

Share Run

```c
#include <stdio.h>

void roundRobin(int burstTime[], int n, int quantum) {
    int remainingBurstTime[n];
    for (int i = 0; i < n; i++) {
        remainingBurstTime[i] = burstTime[i];
    }

    int time = 0;
    while (1) {
        int done = 1;
        for (int i = 0; i < n; i++) {
            if (remainingBurstTime[i] > 0) {
                done = 0;
                if (remainingBurstTime[i] > quantum) {
                    remainingBurstTime[i] -= quantum;
                    time += quantum;
                } else {
                    time += remainingBurstTime[i];
                    remainingBurstTime[i] = 0;
                }
                printf("Process %d executed at time %d\n", i+1,
                    time);
            }
        }
        if (done) break;
    }
}
```

Output

Clear

```
Process 1 executed at time 3
Process 2 executed at time 6
Process 3 executed at time 9
Process 1 executed at time 12
Process 2 executed at time 15
Process 3 executed at time 18
Process 2 executed at time 20
Process 3 executed at time 21


=== Code Execution Successful ===
```

C Online Compiler

main.c

Share Run

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

int main() {
    key_t key = ftok("shmfile", 65);
    int shmid = shmget(key, 1024, 0666 | IPC_CREAT);
    char *str = (char*) shmat(shmid, (void*)0, 0);

    printf("Write Data: ");
    gets(str);

    printf("Data written in memory: %s\n", str);

    shmdt(str);

    return 0;
}
```

Output

Clear

```
Process 1 executed at time 3
Process 2 executed at time 6
Process 3 executed at time 9
Process 1 executed at time 12
Process 2 executed at time 15
Process 3 executed at time 18
Process 2 executed at time 20
Process 3 executed at time 21


=== Code Execution Successful ===
```

main.c

Share Run

Output

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

struct message {
    long mtype;
    char mtext[100];
};

int main() {
    key_t key = ftok("msgqueue", 65);
    int msgid = msgget(key, 0666 | IPC_CREAT);

    struct message msg;
    msg.mtype = 1;

    printf("Enter message: ");
    fgets(msg.mtext, sizeof(msg.mtext), stdin);

    msgsnd(msgid, &msg, sizeof(msg), 0);
    printf("Message sent: %s\n", msg.mtext);

    return 0;
}
```

```
Enter message:
```

main.c

Share Run

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

void* printMessage(void* arg) {
    printf("Hello from thread!\n");
    return NULL;
}

int main() {
    pthread_t thread;

    pthread_create(&thread, NULL, printMessage, NULL);
    pthread_join(thread, NULL);

    return 0;
}
```

Output

```
Hello from thread!


=== Code Execution Successful ===
```

main.c

Share Run

```c
#include <stdio.h>
#include <pthread.h>
#include <unistd.h>

#define N 5

pthread_mutex_t mutex;
pthread_mutex_t chopsticks[N];

void* philosopher(void* num) {
    int id = *(int*)num;

    while (1) {
        printf("Philosopher %d is thinking\n", id);
        usleep(1000);  // Thinking

        pthread_mutex_lock(&chopsticks[id]);
        pthread_mutex_lock(&chopsticks[(id + 1) % N]);

        printf("Philosopher %d is eating\n", id);
        usleep(1000);  // Eating

        pthread_mutex_unlock(&chopsticks[id]);
        pthread_mutex_unlock(&chopsticks[(id + 1) % N]);
    }
}

int main() {
```

Output

```
Philosopher 0 is thinking
Philosopher 2 is thinking
Philosopher 3 is thinking
Philosopher 1 is thinking
Philosopher 4 is thinking
Philosopher 0 is eating
Philosopher 2 is eating
Philosopher 0 is thinking
Philosopher 4 is eating
Philosopher 2 is thinking
Philosopher 1 is eating
Philosopher 4 is thinking
Philosopher 1 is thinking
Philosopher 3 is eating
Philosopher 0 is eating
Philosopher 0 is thinking
Philosopher 2 is eating
Philosopher 4 is eating
Philosopher 3 is thinking
Philosopher 4 is thinking
Philosopher 3 is eating
Philosopher 1 is eating
Philosopher 2 is thinking
Philosopher 1 is thinking
Philosopher 3 is thinking
Philosopher 0 is eating
Philosopher 2 is eating
Philosopher 4 is eating
```

main.c

Share Run

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

void worstFit(int blockSize[], int m, int processSize[], int n) {
    int allocation[n];
    for (int i = 0; i < n; i++) {
        allocation[i] = -1;
    }

    for (int i = 0; i < n; i++) {
        int worstIdx = -1;
        for (int j = 0; j < m; j++) {
            if (blockSize[j] >= processSize[i]) {
                if (worstIdx == -1 || blockSize[j] >
                    blockSize[worstIdx]) {
                    worstIdx = j;
                }
            }
        }

        if (worstIdx != -1) {
            allocation[i] = worstIdx;
            blockSize[worstIdx] -= processSize[i];
        }
    }

    for (int i = 0; i < n; i++) {
        if (allocation[i] != -1) {
```

Output Clear

```
Process 1 allocated to block 5
Process 2 allocated to block 2
Process 3 allocated to block 5
Process 4 not allocated


=== Code Execution Successful ===
```

main.c

Share Run

```c
#include <stdio.h>
#include <dirent.h>

int main() {
    struct dirent *entry;
    DIR *dp = opendir(".");

    if (dp == NULL) {
        perror("opendir");
        return 1;
    }

    while ((entry = readdir(dp)) != NULL) {
        printf("%s\n", entry->d_name);
    }

    closedir(dp);
    return 0;
}
```

Output

```
.
..
.bash_logout
.bashrc
.profile


=== Code Execution Successful ===
```

main.c

Share Run

```c
#include <stdio.h>
#include <dirent.h>
#include <string.h>

void listDirectory(char *path) {
    struct dirent *entry;
    DIR *dp = opendir(path);

    if (dp == NULL) {
        perror("opendir");
        return;
    }

    while ((entry = readdir(dp)) != NULL) {
        printf("%s/%s\n", path, entry->d_name);
    }

    closedir(dp);
}

int main() {
    char *dirPath = ".";
    listDirectory(dirPath);
    return 0;
}
```

Output

```
./.
./..
./.bash_logout
./.bashrc
./.profile


=== Code Execution Successful ===
```

main.c

Share Run

```c
#include <stdio.h>

struct Employee {
    int empId;
    char name[30];
};

int main() {
    FILE *fp = fopen("employee.dat", "rb+");
    if (fp == NULL) {
        printf("File could not be opened\n");
        return 1;
    }

    struct Employee emp = {101, "John Doe"};
    fseek(fp, 0, SEEK_SET);
    fwrite(&emp, sizeof(struct Employee), 1, fp);

    fseek(fp, 0, SEEK_SET);
    fread(&emp, sizeof(struct Employee), 1, fp);

    printf("Employee ID: %d\n", emp.empId);
    printf("Employee Name: %s\n", emp.name);

    fclose(fp);
    return 0;
}
```

Output

```
File could not be opened


=== Code Exited With Errors ===
```