Here are the errors and suggestions for improvement in the code:

Memory leak: In the main function, you are allocating memory for processes using malloc and then reassigning it using realloc. This will cause a memory leak because the original memory block allocated by malloc is not freed. To fix this, you should free the original memory block before reassigning it.

Incorrect calculation of average waiting time and turnaround time: In the FCFS function, you are calculating the average waiting time and turnaround time using integer division, which will truncate the decimal part. To fix this, you should use floating-point division.

No error checking for memory allocation: In the main function, you are not checking if the memory allocation using malloc and realloc is successful. To fix this, you should check the return value of these functions and handle errors accordingly.

No sorting of processes based on arrival time: In the FCFS function, you are not sorting the processes based on their arrival time. To fix this, you should add a sorting algorithm to sort the processes before calculating the waiting time and turnaround time.

===========

#include <stdio.h>

#include <stdlib.h>

struct Process {

char name;

int arrival\_time, burst\_time, waiting\_time, turnaround\_time;

} \*processes;

void FCFS(int n) {

// Sort the processes by arrival time

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

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

if (processes[i].arrival\_time > processes[j].arrival\_time) {

struct Process temp = processes[i];

processes[i] = processes[j];

processes[j] = temp;

}

}

}

// Calculate waiting time and turnaround time

processes[0].waiting\_time = 0;

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

processes[i].waiting\_time = processes[i - 1].burst\_time + processes[i - 1].waiting\_time;

}

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

processes[i].turnaround\_time = processes[i].burst\_time + processes[i].waiting\_time;

}

double avg\_waiting\_time = 0, avg\_turnaround\_time = 0;

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

avg\_waiting\_time += processes[i].waiting\_time;

avg\_turnaround\_time += processes[i].turnaround\_time;

printf("Process %c - Waiting Time: %d - Turnaround Time: %d\n", processes[i].name, processes[i].waiting\_time, processes[i].turnaround\_time);

}

avg\_waiting\_time /= n;

avg\_turnaround\_time /= n;

printf("Average Waiting Time: %.2f\n", avg\_waiting\_time);

printf("Average Turnaround Time: %.2f\n", avg\_turnaround\_time);

}

int main() {

int n;

printf("Enter number of processes: ");

scanf("%d", &n);

processes = (struct Process \*)malloc(n \* sizeof(struct Process));

if (processes == NULL) {

printf("Memory allocation failed\n");

return 1;

}

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

printf("Enter details for process %d\n", i + 1);

printf("Enter name: ");

scanf(" %c", &processes[i].name);

printf("Enter arrival time: ");

scanf("%d", &processes[i].arrival\_time);

printf("Enter burst time: ");

scanf("%d", &processes[i].burst\_time);

}

FCFS(n);

// Add another process

n++;

processes = (struct Process \*)realloc(processes, n \* sizeof(struct Process));

if (processes == NULL) {

printf("Memory allocation failed\n");

return 1;

}

printf("Enter details for the additional process\n");

printf("Enter name: ");

scanf(" %c", &processes[n - 1].name);

printf("Enter arrival time: ");

scanf("%d", &processes[n - 1].arrival\_time);

printf("Enter burst time: ");

scanf("%d", &processes[n - 1].burst\_time);

FCFS(n);

free(processes);

return 0;

}

===========

Fixed memory leak: We added a free(processes) statement to release the memory allocated by malloc and realloc when we're done using it.

Corrected calculation of average waiting time and turnaround time: We changed the data type of avg\_waiting\_time and avg\_turnaround\_time to double to allow for decimal calculations. We also used floating-point division (/=) to calculate the averages.

Added error checking for memory allocation: We added checks to ensure that the memory allocation using malloc and realloc was successful. If the allocation fails, we print an error message and exit the program.

Added sorting of processes based on arrival time: We added a sorting algorithm to sort the processes based on their arrival time. This is necessary for the FCFS scheduling algorithm to work correctly.

========================

The struct Process defines a structure to represent a process, with fields for the process name, arrival time, burst time, waiting time, and turnaround time.

The FCFS function calculates the waiting time and turnaround time for each process using the FCFS scheduling algorithm. It sorts the processes based on their arrival time, calculates the waiting time and turnaround time for each process, and then calculates the average waiting time and turnaround time.

In the main function, we prompt the user to enter the number of processes and then allocate memory for the processes array using malloc. We then loop through each process, prompting the user to enter the process details (name, arrival time, and burst time).

We call the FCFS function to calculate the waiting time and turnaround time for each process.

We then add another process to the array using realloc and prompt the user to enter the details for the additional process.

We call the FCFS function again to recalculate the waiting time and turnaround time for each process, including the additional process.

Finally, we free the memory allocated for the processes array using free

=============

FCFS Function

Sorts the processes based on their arrival time using a bubble sort algorithm

Calculates the waiting time and turnaround time for each process using the FCFS scheduling algorithm

Calculates the average waiting time and turnaround time

Main Function

Prompts the user to enter the number of processes

Allocates memory for the processes array using malloc

Loops through each process and prompts the user to enter details (name, arrival time, and burst time)

Calls the FCFS function to calculate waiting time and turnaround time

Adds another process to the array using realloc

Calls the FCFS function again to recalculate waiting time and turnaround time

Frees the memory allocated for the processes array using free