

Project-report: scheduling algorithm

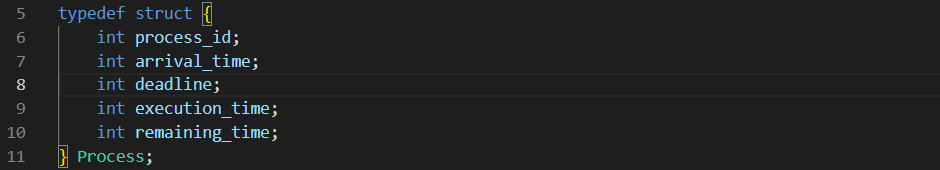


Abstract

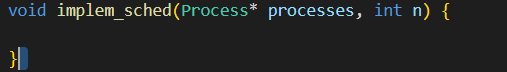
Scheduling algorithms are methods used in operating systems to distribute resources among processes in a scheduled manner. The purpose of these algorithms is to minimize resource starvation by ensuring that each process gets its required CPU time.

The algorithm implemented here is a **priority driven** algorithm i.e., a process is assigned a priority , and the scheduler selects the process to run according to it . In this case, the priority is set according to the deadline of the processes , and in case of having same deadlines, it selects processes based on their shortest execution time. The integration of two approaches makes it an efficient solution for real – time environment.

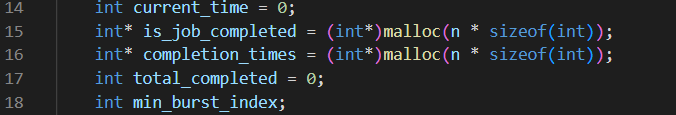
Implementation



This structure represent a process with attributes like process ID , arrival time , deadline , execution time , and remaining execution time.



This function takes an array of processes and its size is n as input and schedules tasks using the above logic.



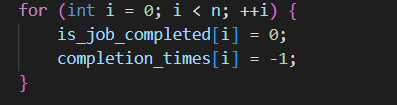
*current time* :- keeps track of the current time during scheduling.

*is\_job\_completed* :- is an array to mark whether a process is completed or not.

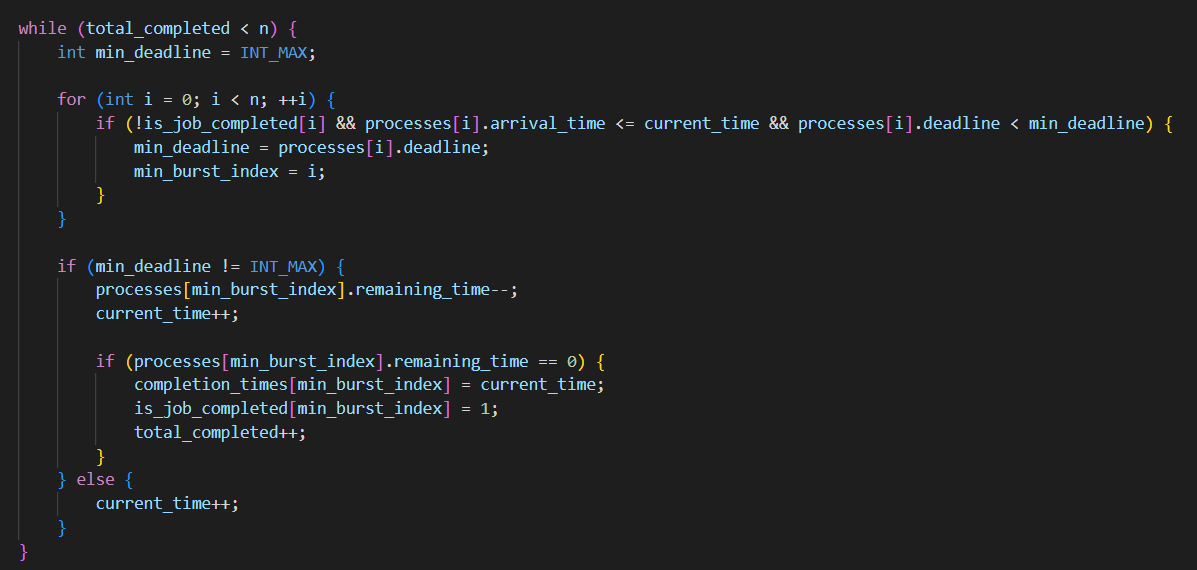
*completion\_time* :- stores the completion time of each process.

*total\_completed* :- keeps track of the total number of completed processes.

*min\_burst\_index* :- keeps track of the process with the shortest remaining time among processes with the same deadline.



Initilize the arrays *is\_job\_completed* and *completed\_times* to their initial values.

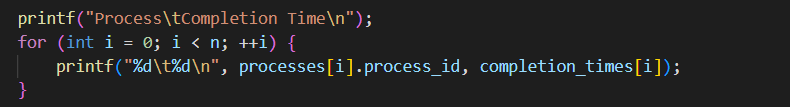


The outer loop continues till all processes are completed (*total\_completed* reaches n).

In the inner loop , it find the processes with the earliest deadline (*min\_deadline)* and the shortest remaining time. If a suitable process is found , it reduces its remaining time by and increments the current time.

If the process is completed execution (*remaining\_time* becomes 0) , its completion time is recorded and it is marked as complete.

If no process is found with a deadline before the current time , the current time is simply incremented.



Finally the completion time of each task is printed .

This implementation ensures the processes are executed based on their deadlines and , within processes with the same deadline , the shortest job is executed first.

CODE

#include <stdio.h>

#include <stdlib.h>

#include <limits.h>

typedef struct {

    int process\_id;

    int arrival\_time;

    int deadline;

    int execution\_time;

    int remaining\_time;

} Process;

void implem\_sched(Process\* processes, int n) {

    int current\_time = 0;

    int\* is\_job\_completed = (int\*)malloc(n \* sizeof(int));

    int\* completion\_times = (int\*)malloc(n \* sizeof(int));

    int total\_completed = 0;

    int min\_burst\_index;

    for (int i = 0; i < n; ++i) {

        is\_job\_completed[i] = 0;

        completion\_times[i] = -1;

    }

    while (total\_completed < n) {

        int min\_deadline = INT\_MAX;

        for (int i = 0; i < n; ++i) {

            if (!is\_job\_completed[i] && processes[i].arrival\_time <= current\_time && processes[i].deadline < min\_deadline) {

                min\_deadline = processes[i].deadline;

                min\_burst\_index = i;

            }

        }

        if (min\_deadline != INT\_MAX) {

            processes[min\_burst\_index].remaining\_time--;

            current\_time++;

            if (processes[min\_burst\_index].remaining\_time == 0) {

                completion\_times[min\_burst\_index] = current\_time;

                is\_job\_completed[min\_burst\_index] = 1;

                total\_completed++;

            }

        } else {

            current\_time++;

        }

    }

    printf("Process\tCompletion Time\n");

    for (int i = 0; i < n; ++i) {

        printf("%d\t%d\n", processes[i].process\_id, completion\_times[i]);

    }

    free(is\_job\_completed);

    free(completion\_times);

}

int main() {

    int n = 5;  // Number of processes

    Process processes[] = {

        {1, 0, 10, 10, 10},

        {2, 1, 6, 8, 8},

        {3, 2, 8, 6, 6},

        {4, 3, 12, 4, 4},

        {5, 4, 15, 5, 5}

    };

    implem\_sched(processes, n);

    return 0;

}

OUTPUT

