CSE 321: Operating Systems

Lab Assignment 4

Name: Mollah Md Saif

ID: 20101416

Section: 03

**Task 1**

#include<stdio.h>

#include<stdbool.h>

#define MAX\_BURST\_TIME 1000000

struct process {

int pid;

int arrival\_time;

int const\_burst\_time;

int burst\_time;

int waiting\_time;

int turnaround\_time;

};

int main() {

int n;

printf("Enter number of processes: ");

scanf("%d", &n);

printf("\n");

struct process processes[n];

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

printf("Enter process %d \"process\_id arrival\_time burst\_time\": ", i + 1);

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

processes[i].const\_burst\_time = processes[i].burst\_time;

processes[i].waiting\_time = 0;

processes[i].turnaround\_time = 0;

}

bool is\_running[n];

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

is\_running[i] = false;

}

bool is\_completed[n];

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

is\_completed[i] = false;

}

int time = 0;

while(true)

{

//debug

/\*

printf("Time: %d\n", time);

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

printf("Process id: %d, burst time: %d, waiting time: %d, is running: %d, is completed: %d\n", processes[i].pid, processes[i].burst\_time, processes[i].waiting\_time, is\_running[i], is\_completed[i]);

}

printf("\n");

\*/

// check arrived processes

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

if (!is\_completed[i] && !is\_running[i] && processes[i].arrival\_time <= time) {

is\_running[i] = true;

}

}

// get shortest burst time of running processes

int shortest\_burst\_time = MAX\_BURST\_TIME;

int shortest\_burst\_time\_index = -1;

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

if (is\_running[i] && processes[i].burst\_time < shortest\_burst\_time) {

shortest\_burst\_time = processes[i].burst\_time;

shortest\_burst\_time\_index = i;

}

}

// run shortest burst time process

if (shortest\_burst\_time\_index != -1) {

processes[shortest\_burst\_time\_index].burst\_time--;

if (processes[shortest\_burst\_time\_index].burst\_time == 0) {

is\_running[shortest\_burst\_time\_index] = false;

is\_completed[shortest\_burst\_time\_index] = true;

processes[shortest\_burst\_time\_index].turnaround\_time = processes[shortest\_burst\_time\_index].waiting\_time + processes[shortest\_burst\_time\_index].const\_burst\_time;

}

}

// calculate waiting time

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

if (i != shortest\_burst\_time\_index && is\_running[i]) {

processes[i].waiting\_time++;

}

}

// finished all processes

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

if (!is\_completed[i]) {

goto not\_finished;

}

}

break;

not\_finished:

time++;

continue;

}

// print table

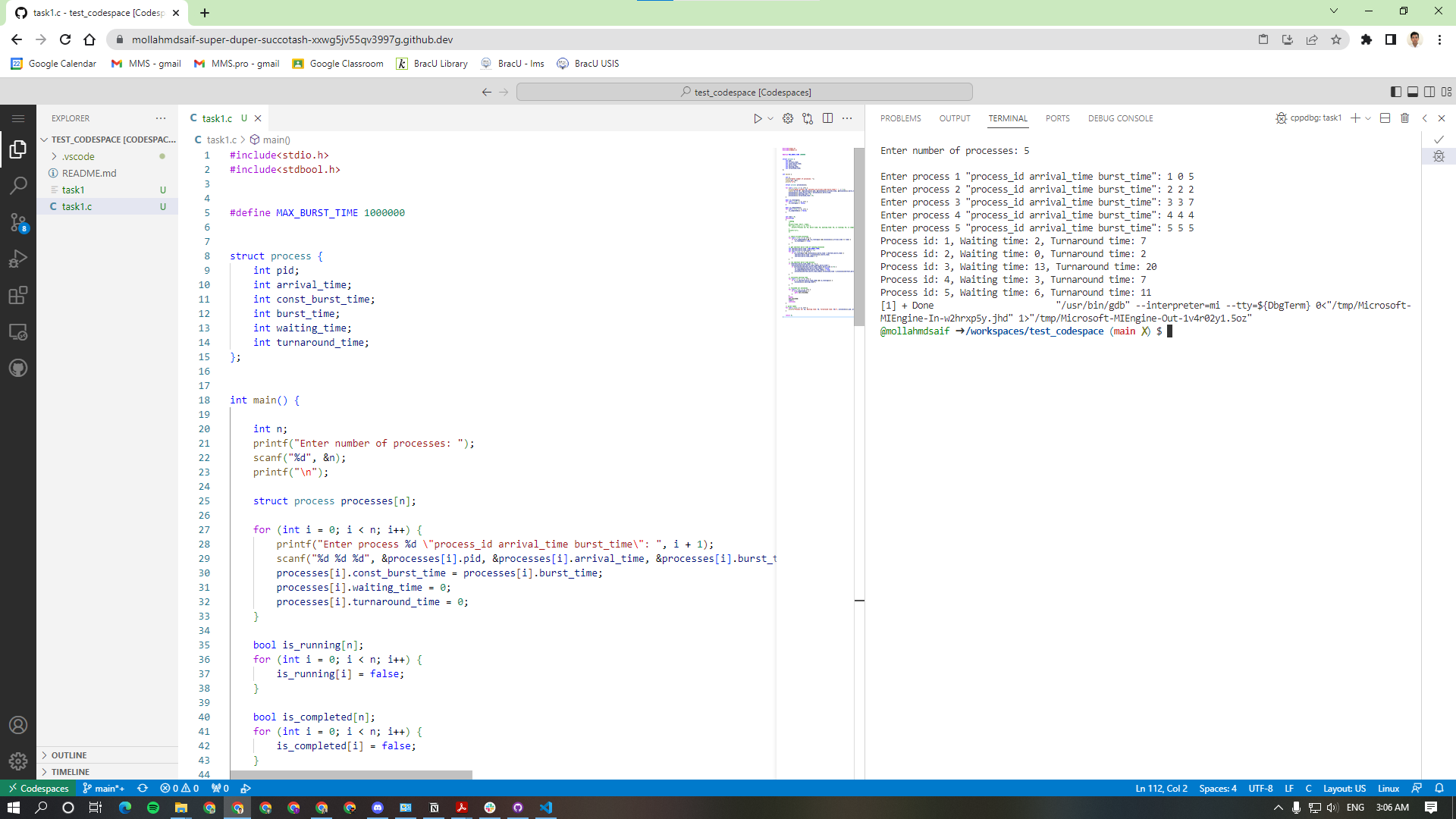
for (int i = 0; i < n; i++) {

printf("Process id: %d, Waiting time: %d, Turnaround time: %d\n", processes[i].pid, processes[i].waiting\_time, processes[i].turnaround\_time);

}

return 0;

}



**Task 2**

#include<stdio.h>

#include<stdbool.h>

struct process {

int pid;

int const\_burst\_time;

int burst\_time;

int waiting\_time;

int turnaround\_time;

};

int main() {

int time\_quantum;

printf("Enter time quantum: ");

scanf("%d", &time\_quantum);

printf("\n");

int n;

printf("Enter number of processes: ");

scanf("%d", &n);

printf("\n");

struct process processes[n];

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

printf("Enter process %d \"process\_id burst\_time\": ", i + 1);

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

processes[i].const\_burst\_time = processes[i].burst\_time;

processes[i].waiting\_time = 0;

processes[i].turnaround\_time = 0;

}

bool is\_running[n];

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

is\_running[i] = true;

}

bool is\_completed[n];

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

is\_completed[i] = false;

}

int time = 0;

int idx = 0;

while(true)

{

// run process

if (!is\_completed[idx] && is\_running[idx]) {

if (processes[idx].burst\_time >= time\_quantum)

{

processes[idx].burst\_time -= time\_quantum;

time += time\_quantum;

} else {

time += processes[idx].burst\_time;

processes[idx].burst\_time -= processes[idx].burst\_time;

}

}

// check if any process is completed

if (!is\_completed[idx] && processes[idx].burst\_time == 0) {

is\_running[idx] = false;

is\_completed[idx] = true;

processes[idx].turnaround\_time = time;

}

// increment index

idx = (idx + 1) % n;

// finished all processes

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

if (!is\_completed[i]) {

goto not\_finished;

}

}

break;

not\_finished:

continue;

}

// calculate waiting time

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

processes[i].waiting\_time = processes[i].turnaround\_time - processes[i].const\_burst\_time;

}

// print table

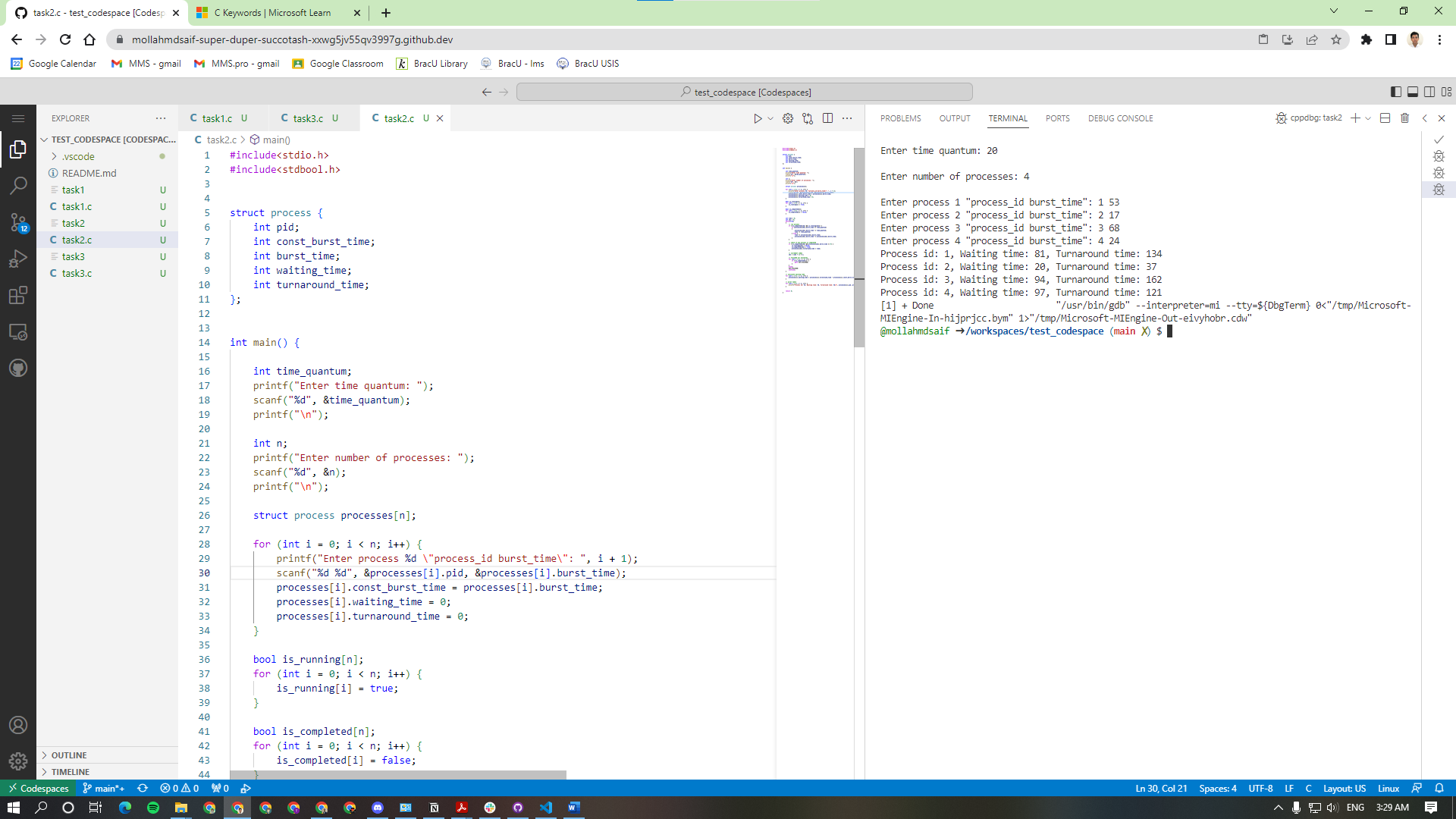
for (int i = 0; i < n; i++) {

printf("Process id: %d, Waiting time: %d, Turnaround time: %d\n", processes[i].pid, processes[i].waiting\_time, processes[i].turnaround\_time);

}

return 0;

}



**Task 3**

#include<stdio.h>

#include<stdbool.h>

#define MAX\_BURST\_TIME 1000000

#define LOWEST\_PRIORITY 1000000

struct process {

int pid;

int arrival\_time;

int const\_burst\_time;

int burst\_time;

int priority;

int waiting\_time;

int turnaround\_time;

};

int main() {

int n;

printf("Enter number of processes: ");

scanf("%d", &n);

printf("\n");

struct process processes[n];

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

printf("Enter process %d \"process\_id arrival\_time burst\_time priority\": ", i + 1);

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

processes[i].const\_burst\_time = processes[i].burst\_time;

processes[i].waiting\_time = 0;

processes[i].turnaround\_time = 0;

}

bool is\_running[n];

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

is\_running[i] = false;

}

bool is\_completed[n];

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

is\_completed[i] = false;

}

int time = 0;

while(true)

{

//debug

/\*

printf("Time: %d\n", time);

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

printf("Process id: %d, burst time: %d, waiting time: %d, is running: %d, is completed: %d\n", processes[i].pid, processes[i].burst\_time, processes[i].waiting\_time, is\_running[i], is\_completed[i]);

}

printf("\n");

\*/

// check arrived processes

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

if (!is\_completed[i] && !is\_running[i] && processes[i].arrival\_time <= time) {

is\_running[i] = true;

}

}

// get highest proirity of running processes

int highest\_priority = LOWEST\_PRIORITY;

int highest\_priority\_index = -1;

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

if (is\_running[i] && processes[i].priority < highest\_priority) {

highest\_priority = processes[i].priority;

highest\_priority\_index = i;

}

}

// run shortest burst time process

if (highest\_priority\_index != -1) {

processes[highest\_priority\_index].burst\_time--;

if (processes[highest\_priority\_index].burst\_time == 0) {

is\_running[highest\_priority\_index] = false;

is\_completed[highest\_priority\_index] = true;

processes[highest\_priority\_index].turnaround\_time = processes[highest\_priority\_index].waiting\_time + processes[highest\_priority\_index].const\_burst\_time;

}

}

// calculate waiting time

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

if (i != highest\_priority\_index && is\_running[i]) {

processes[i].waiting\_time++;

}

}

// finished all processes

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

if (!is\_completed[i]) {

goto not\_finished;

}

}

break;

not\_finished:

time++;

continue;

}

// print table

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

printf("Process id: %d, Waiting time: %d, Turnaround time: %d\n", processes[i].pid, processes[i].waiting\_time, processes[i].turnaround\_time);

}

return 0;

}

