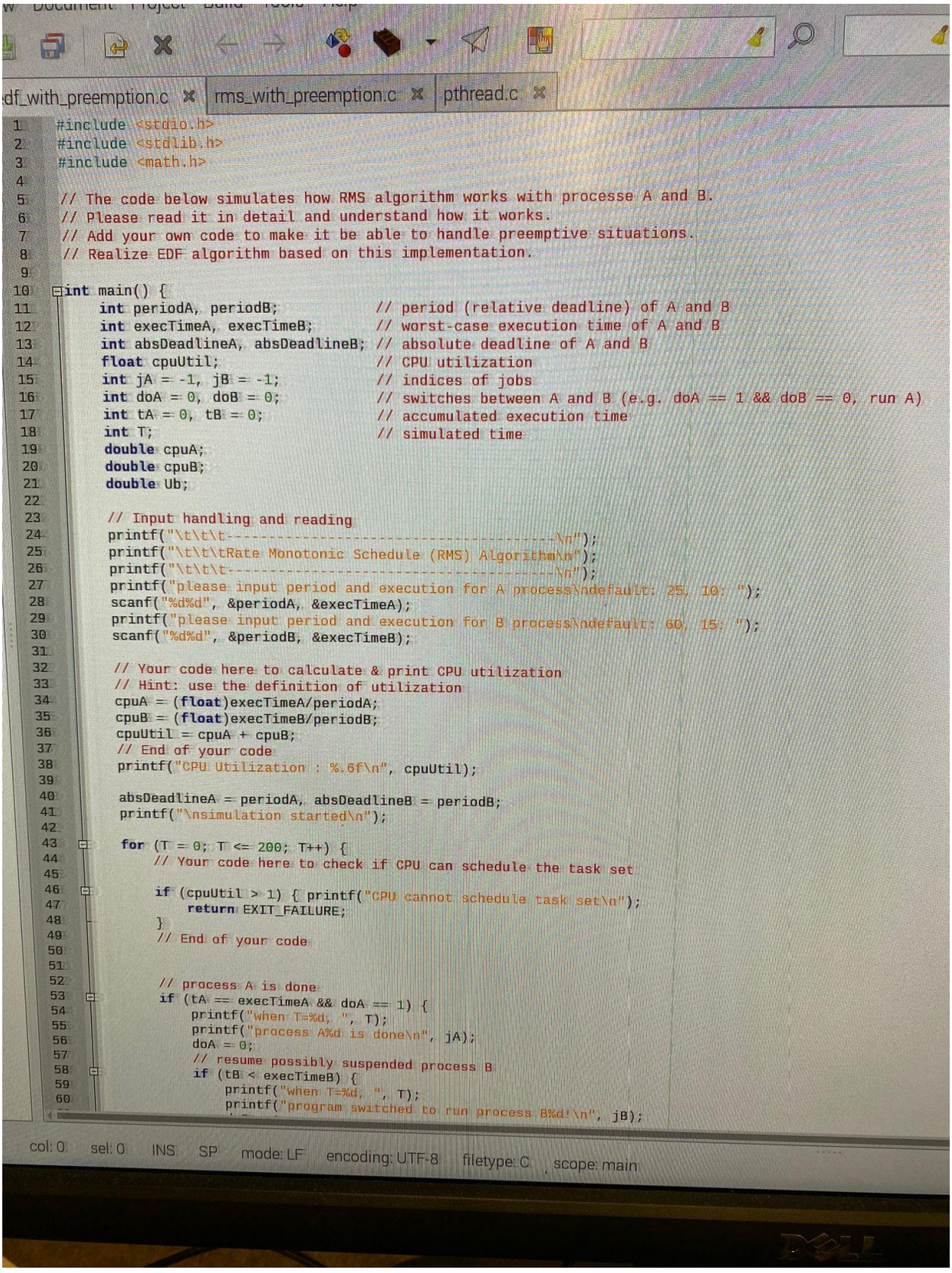
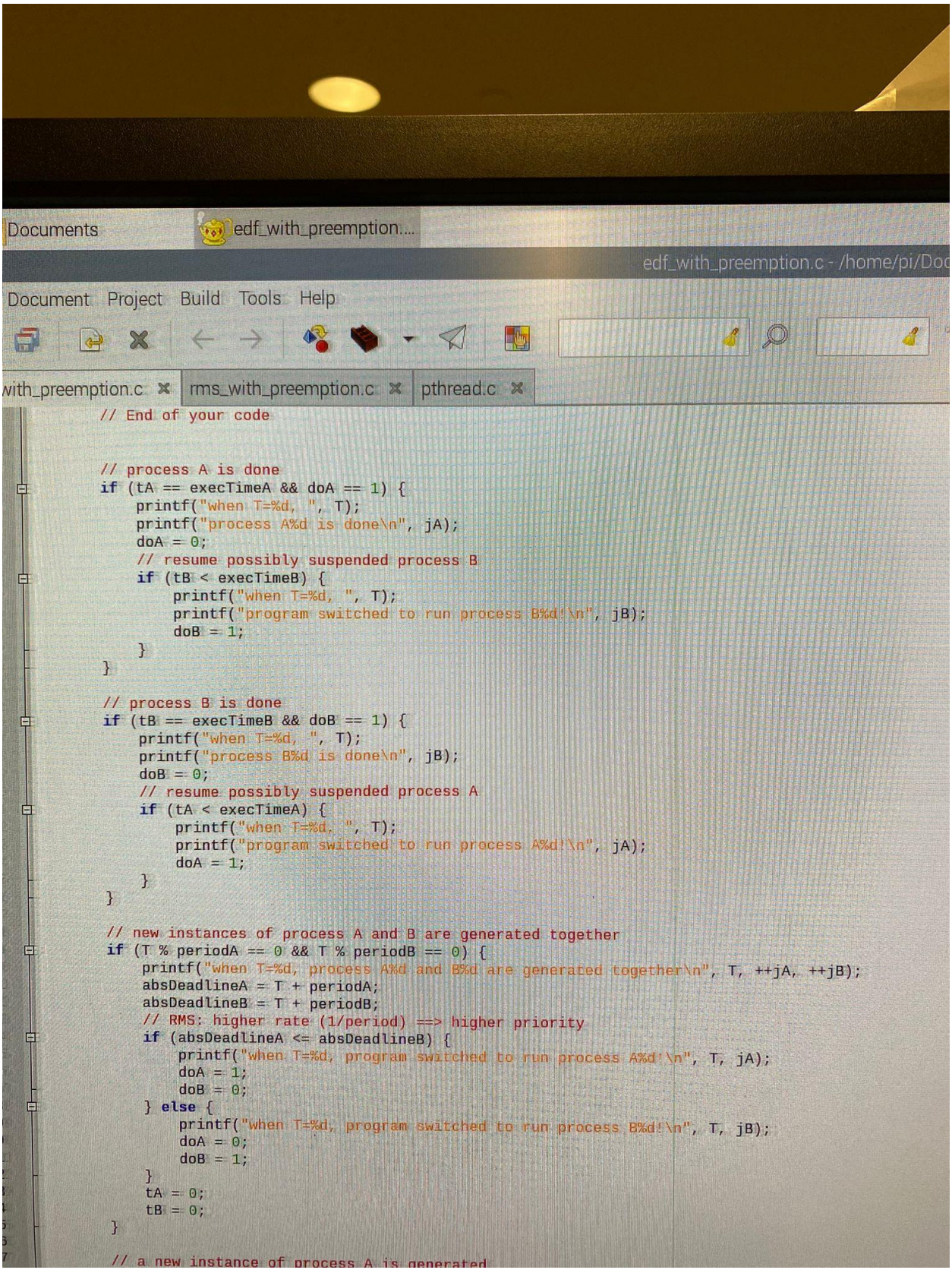
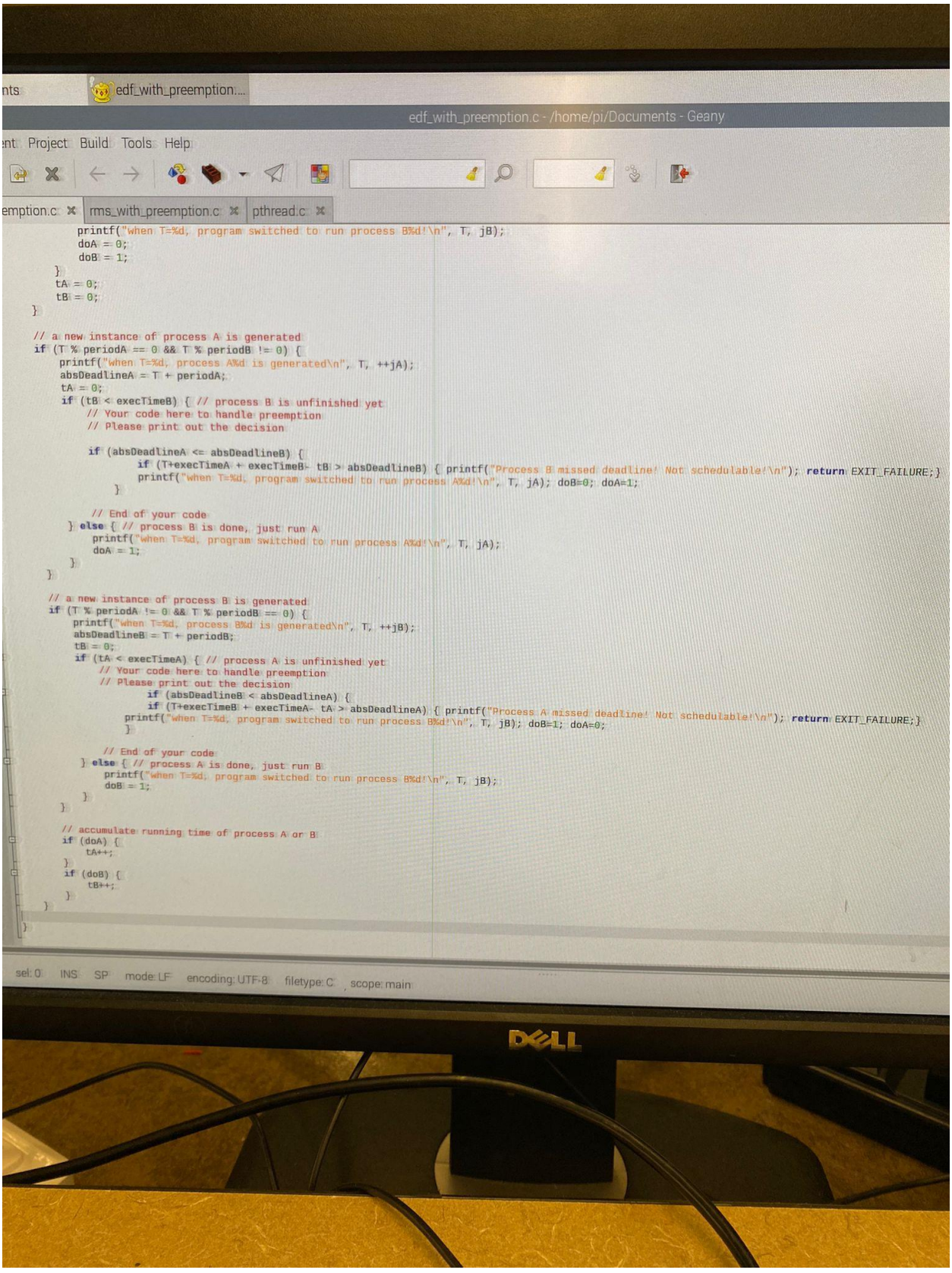


df_with_preemption.c | rms_with_preemption.c | pthread.c

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

// The code below simulates how RMS algorithm works with processe A and B.
// Please read it in detail and understand how it works.
// Add your own code to make it be able to handle preemptive situations.
// Realize EDF algorithm based on this implementation.

int main() {
    int periodA, periodB;           // period (relative deadline) of A and B
    int execTimeA, execTimeB;       // worst-case execution time of A and B
    int absDeadlineA, absDeadlineB; // absolute deadline of A and B
    float cpuUtil;                  // CPU utilization
    int jA = -1, jB = -1;           // indices of jobs
    int doA = 0, doB = 0;           // switches between A and B (e.g. doA == 1 && doB == 0, run A)
    int tA = 0, tB = 0;             // accumulated execution time
    int T;                          // simulated time
    double cpuA;
    double cpuB;
    double Ub;

    // Input handling and reading
    printf("\t\t\t---------------------------------------\n");
    printf("\t\t\tRate Monotonic Schedule (RMS) Algorithm\n");
    printf("\t\t\t---------------------------------------\n");
    printf("please input period and execution for A process\ndefault: 25, 10: ");
    scanf("%d%d", &periodA, &execTimeA);
    printf("please input period and execution for B process\ndefault: 60, 15: ");
    scanf("%d%d", &periodB, &execTimeB);

    // Your code here to calculate & print CPU utilization
    // Hint: use the definition of utilization
    cpuA = (float)execTimeA/periodA;
    cpuB = (float)execTimeB/periodB;
    cpuUtil = cpuA + cpuB;
    // End of your code
    printf("CPU Utilization : %.6f\n", cpuUtil);

    absDeadlineA = periodA, absDeadlineB = periodB;
    printf("\nsimulation started\n");

    for (T = 0; T <= 200; T++) {
        // Your code here to check if CPU can schedule the task set

        if (cpuUtil > 1) { printf("CPU cannot schedule task set\n");
            return EXIT_FAILURE;
        }
        // End of your code


        // process A is done
        if (tA == execTimeA && doA == 1) {
            printf("when T=%d, ", T);
            printf("process A%d is done\n", jA);
            doA = 0;
            // resume possibly suspended process B
            if (tB < execTimeB) {
                printf("when T=%d, ", T);
                printf("program switched to run process B%d!\n", jB);
```

col: 0 sel: 0 INS SP mode: LF encoding: UTF-8 filetype: C scope: main


Documents
edf_with_preemption....
edf_with_preemption.c - /home/pi/Doc
Document Project Build Tools Help
with_preemption.c
rms_with_preemption.c
pthread.c
// End of your code
// process A is done
if (tA == execTimeA && doA == 1) {
printf("when T=%d, ", T);
printf("process A%d is done\n", jA);
doA = 0;
// resume possibly suspended process B
if (tB < execTimeB) {
printf("when T=%d, ", T);
printf("program switched to run process B%d!\n", jB);
doB = 1;
}
}
// process B is done
if (tB == execTimeB && doB == 1) {
printf("when T=%d, ", T);
printf("process B%d is done\n", jB);
doB = 0;
// resume possibly suspended process A
if (tA < execTimeA) {
printf("when T=%d, ", T);
printf("program switched to run process A%d!\n", jA);
doA = 1;
}
}
// new instances of process A and B are generated together
if (T % periodA == 0 && T % periodB == 0) {
printf("when T=%d, process A%d and B%d are generated together\n", T, ++jA, ++jB);
absDeadlineA = T + periodA;
absDeadlineB = T + periodB;
// RMS: higher rate (1/period) ==> higher priority
if (absDeadlineA <= absDeadlineB) {
printf("when T=%d, program switched to run process A%d!\n", T, jA);
doA = 1;
doB = 0;
} else {
printf("when T=%d, program switched to run process B%d!\n", T, jB);
doA = 0;
doB = 1;
}
tA = 0;
tB = 0;
}
// a new instance of process A is generated


edf_with_preemption....
edf_with_preemption.c - /home/pi/Documents - Geany
Project Build Tools Help
emption.c
rms_with_preemption.c
pthread.c
printf("when T=%d, program switched to run process B%d!\n", T, jB);
doA = 0;
doB = 1;
tA = 0;
tB = 0;
// a new instance of process A is generated
if (T % periodA == 0 && T % periodB != 0) {
printf("when T=%d, process A%d is generated\n", T, ++jA);
absDeadlineA = T + periodA;
tA = 0;
if (tB < execTimeB) { // process B is unfinished yet
// Your code here to handle preemption
// Please print out the decision
if (absDeadlineA <= absDeadlineB) {
if (T+execTimeA + execTimeB- tB > absDeadlineB) { printf("Process B missed deadline! Not schedulable!\n"); return EXIT_FAILURE;}
printf("when T=%d, program switched to run process A%d!\n", T, jA); doB=0; doA=1;
}
// End of your code
} else { // process B is done, just run A
printf("when T=%d, program switched to run process A%d!\n", T, jA);
doA = 1;
}
}
// a new instance of process B is generated
if (T % periodA != 0 && T % periodB == 0) {
printf("when T=%d, process B%d is generated\n", T, ++jB);
absDeadlineB = T + periodB;
tB = 0;
if (tA < execTimeA) { // process A is unfinished yet
// Your code here to handle preemption
// Please print out the decision
if (absDeadlineB < absDeadlineA) {
if (T+execTimeB + execTimeA- tA > absDeadlineA) { printf("Process A missed deadline! Not schedulable!\n"); return EXIT_FAILURE;}
printf("when T=%d, program switched to run process B%d!\n", T, jB); doB=1; doA=0;
}
// End of your code
} else { // process A is done, just run B
printf("when T=%d, program switched to run process B%d!\n", T, jB);
doB = 1;
}
}
// accumulate running time of process A or B
if (doA) {
tA++;
}
if (doB) {
tB++;
}
}
}
sel: 0 INS SP mode: LF encoding: UTF-8 filetype: C scope: main
DELL