**ASSIGNMENT 3**

/**/Shortest Job First (Preemptive)**

#include <stdio.h>

#include <stdbool.h>

struct Process

{

int pid;

int at;

int bt;

int ct, tt, wt, rt, st;

};

int main()

{

int size = 0;

printf("Enter number of processes: ");

scanf("%d", &size);

struct Process ps[size];

printf("\nEnter process Details: \n");

for (int i = 0; i < size; ++i)

{

printf("Enter %dth process details: \n", i + 1);

ps[i].pid = i + 1;

printf("\tEnter Arrival Time: ");

scanf("%d", &ps[i].at);

printf("\tEnter Burst Time: ");

scanf("%d", &ps[i].bt);

}

int n = size;

int completed = 0;

int currentTime = 0;

int burstTimeR[4];

bool iscompleted[4] = {false};

float avgWT = 0, avgTT = 0, avgRT = 0;

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

{

burstTimeR[i] = ps[i].bt;

}

while (completed != n)

{

int minimum = 99999;

int miniI = -1;

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

{

if ((ps[i].at <= currentTime) && (iscompleted[i] == false))

{

if (burstTimeR[i] < minimum)

{

minimum = burstTimeR[i];

miniI = i;

}

if (burstTimeR[i] == minimum)

{

if (ps[i].at < ps[miniI].at)

{

minimum = burstTimeR[i];

miniI = i;

}

}

}

}

if (miniI == -1)

{

currentTime++;

}

else

{

if (burstTimeR[miniI] == ps[miniI].bt)

{

ps[miniI].st = currentTime;

}

burstTimeR[miniI] -= 1;

currentTime++;

if (burstTimeR[miniI] == 0)

{

ps[miniI].ct = currentTime;

ps[miniI].tt = ps[miniI].ct - ps[miniI].at;

ps[miniI].wt = ps[miniI].tt - ps[miniI].bt;

ps[miniI].rt = ps[miniI].st - ps[miniI].at;

avgWT += ps[miniI].wt;

avgTT += ps[miniI].tt;

avgRT += ps[miniI].rt;

completed++;

iscompleted[miniI] = true;

}

}

}

printf("\n\n====================================================================================\n");

printf("PID \t AT \t BT \t CT \t TAT \t WT \t RT \t\n");

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

{

printf("%d \t %d \t %d \t %d \t %d \t %d \t %d \t\n", ps[i].pid, ps[i].at, ps[i].bt, ps[i].ct, ps[i].tt, ps[i].wt, ps[i].rt);

}

printf("\n\n====================================================================================\n");

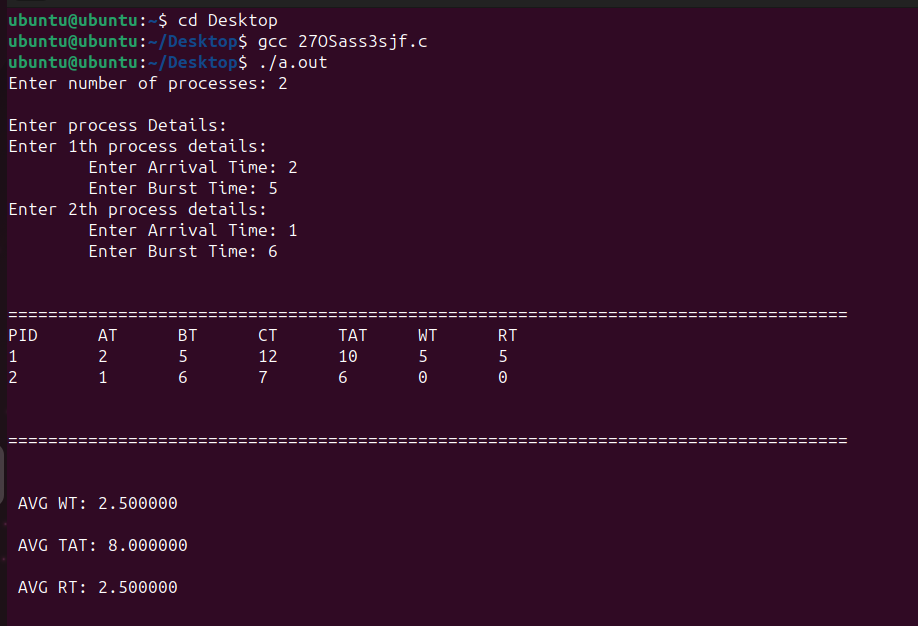
printf("\n\n AVG WT: %f", avgWT / n);

printf("\n\n AVG TAT: %f", avgTT / n);

printf("\n\n AVG RT: %f", avgRT / n);

printf("\n\n====================================================================================\n");

}



**##Round Robin**

#include <stdio.h>

// #include <limits.h>

#include <stdbool.h>

#include <stdlib.h> //for qsort

struct process\_struct

{

int pid;

int at;

int bt;

int ct, wt, tat, rt, start\_time;

int bt\_remaining;

} ps[100];

int findmax(int a, int b)

{

return a > b ? a : b;

}

int comparatorAT(const void \*a, const void \*b)

{

int x = ((struct process\_struct \*)a)->at;

int y = ((struct process\_struct \*)b)->at;

if (x < y)

return -1;

else if (x >= y)

return 1;

}

int comparatorPID(const void \*a, const void \*b)

{

int x = ((struct process\_struct \*)a)->pid;

int y = ((struct process\_struct \*)b)->pid;

if (x < y)

return -1;

else if (x >= y)

return 1;

}

int main()

{

int n, index;

int cpu\_utilization;

bool visited[100] = {false}, is\_first\_process = true;

int current\_time = 0, max\_completion\_time;

int completed = 0, tq, total\_idle\_time = 0, length\_cycle;

printf("Enter total number of processes: ");

scanf("%d", &n);

int queue[100], front = -1, rear = -1;

float sum\_tat = 0, sum\_wt = 0, sum\_rt = 0;

printf("\nEnter process Details: \n");

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

{

printf("Enter %dth process details: \n", i + 1);

ps[i].pid = i + 1;

printf("\tEnter Arrival Time: ");

scanf("%d", &ps[i].at);

ps[i].pid = i;

printf("\tEnter Burst Time: ");

scanf("%d", &ps[i].bt);

ps[i].bt\_remaining = ps[i].bt;

}

printf("\nEnter time quanta: ");

scanf("%d", &tq);

qsort((void \*)ps, n, sizeof(struct process\_struct), comparatorAT);

front = rear = 0;

queue[rear] = 0;

visited[0] = true;

while (completed != n)

{

index = queue[front];

front++;

if (ps[index].bt\_remaining == ps[index].bt)

{

ps[index].start\_time = findmax(current\_time, ps[index].at);

total\_idle\_time += (is\_first\_process == true) ? 0 : ps[index].start\_time - current\_time;

current\_time = ps[index].start\_time;

is\_first\_process = false;

}

if (ps[index].bt\_remaining - tq > 0)

{

ps[index].bt\_remaining -= tq;

current\_time += tq;

}

else

{

current\_time += ps[index].bt\_remaining;

ps[index].bt\_remaining = 0;

completed++;

ps[index].ct = current\_time;

ps[index].tat = ps[index].ct - ps[index].at;

ps[index].wt = ps[index].tat - ps[index].bt;

ps[index].rt = ps[index].start\_time - ps[index].at;

sum\_tat += ps[index].tat;

sum\_wt += ps[index].wt;

sum\_rt += ps[index].rt;

}

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

{

if (ps[i].bt\_remaining > 0 && ps[i].at <= current\_time && visited[i] == false)

{

queue[++rear] = i;

visited[i] = true;

}

}

if (ps[index].bt\_remaining > 0)

queue[++rear] = index;

if (front > rear)

{

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

{

if (ps[i].bt\_remaining > 0)

{

queue[rear++] = i;

visited[i] = true;

break;

}

}

}

}

max\_completion\_time = 1e-9;

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

max\_completion\_time = findmax(max\_completion\_time, ps[i].ct);

length\_cycle = max\_completion\_time - ps[0].at;

cpu\_utilization = (float)(length\_cycle - total\_idle\_time) / length\_cycle;

qsort((void \*)ps, n, sizeof(struct process\_struct), comparatorPID);

printf("\n\n====================================================================================\n");

printf("\nProcess No.\tAT\tBT\tStart Time\tCT\tTAT\tWT\tRT\n");

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

printf("%d\t\t%d\t%d\t\t%d\t%d\t%d\t%d\t%d\n",i+1, ps[i].at, ps[i].bt, ps[i].start\_time, ps[i].ct, ps[i].tat, ps[i].wt, ps[i].rt);

printf("\n");

printf("\n\n====================================================================================\n");

printf("\nAverage Turn Around time= %.2f", (float)sum\_tat / n);

printf("\nAverage Waiting Time= %.2f", (float)sum\_wt / n);

printf("\nAverage Response Time= %.2f\n", (float)sum\_rt / n);

return 0;

}

