Lab 04 – Group 5

Class : it007.019.1 && IT007.019.2

**Members:**

**1.** Hà Huy Hoàng | 22520460

2. Nguyễn Duy Hoàng | 22520467

3. Nguyễn Hoàng Hiệp | 22520452

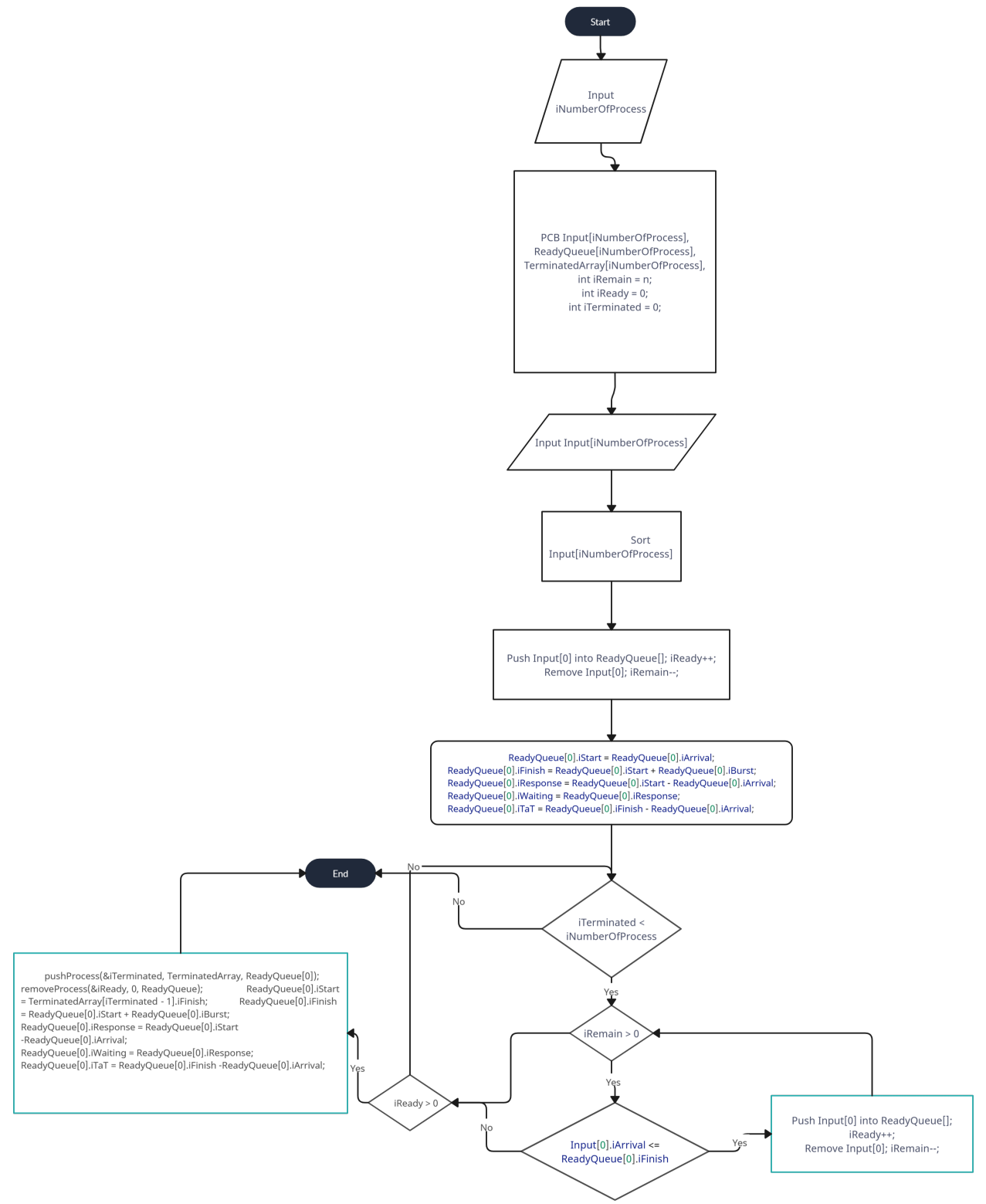
4. Nguyễn Hoàng Phúc | 22521129

**SUMMARY**

|  |  |  |  |
| --- | --- | --- | --- |
| **Task** | | **Status** | **Members** |
| Thực hành | FCFS | Hoàn thành | 3 |
| SJF | Hoàn thành | 2 |
| SRTF | Hoàn thành | 1 |
| RR | Hoàn thành | 4 |

1. **FCFS:**

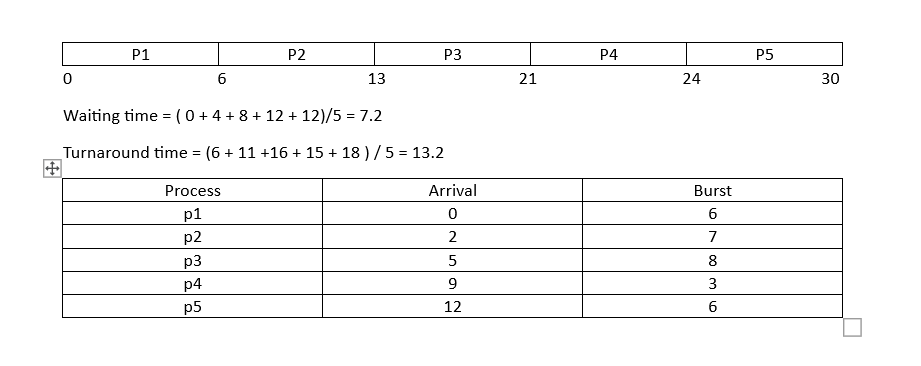
**Lưu đồ giải thuật:**

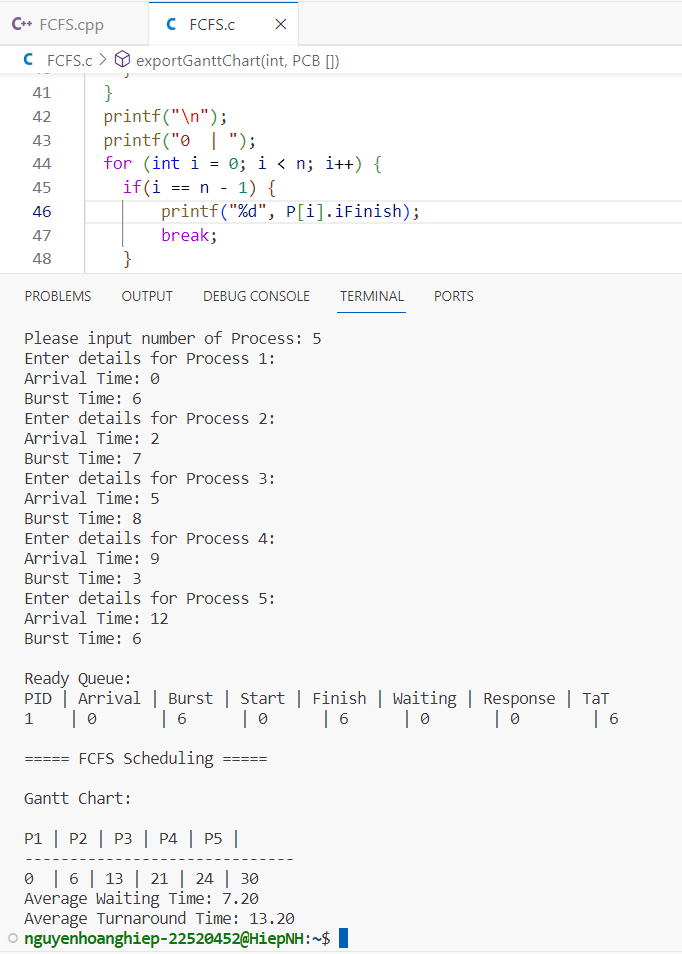


**Code**

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #define SORT\_BY\_ARRIVAL 0  #define SORT\_BY\_PID 1  #define SORT\_BY\_BURST 2  #define SORT\_BY\_START 3  typedef struct{   int iPID;   int iArrival, iBurst;   int iStart, iFinish, iWaiting, iResponse, iTaT;  } PCB;  void inputProcess(int n, PCB P[]) {      for (int i = 0; i < n; ++i) {          printf("Enter details for Process %d:\n", i + 1);          P[i].iPID = i + 1;          printf("Arrival Time: ");          scanf("%d", &P[i].iArrival);          printf("Burst Time: ");          scanf("%d", &P[i].iBurst);      }  }  void printProcess(int n, PCB P[]) {    printf("\nPID | Arrival | Burst | Start | Finish | Waiting | Response | TaT\n");    for (int i = 0; i < n; i++) {      printf("%-4d | %-7d | %-6d | %-6d | %-6d | %-7d | %-8d | %-4d\n", P[i].iPID, P[i].iArrival, P[i].iBurst, P[i].iStart, P[i].iFinish, P[i].iWaiting, P[i].iResponse, P[i].iTaT);    }  }  void exportGanttChart(int n, PCB P[]) {    printf("\nGantt Chart:\n\n");    for (int i = 0; i < n; i++) {      //printf("0 ");      printf("P%d | ", P[i].iPID);    }    printf("\n");    for (int i = 0; i < n; i++) {      for (int j = 0; j < P[i].iBurst; j++) {        printf("-");      }    }    printf("\n");    printf("0  | ");    for (int i = 0; i < n; i++) {      if(i == n - 1) {          printf("%d", P[i].iFinish);          break;      }      printf("%d | ", P[i].iFinish);    }    printf("\n");  }  void pushProcess(int \*n, PCB P[], PCB Q) {    P[\*n] = Q;    (\*n)++;  }  void removeProcess(int \*n, int index, PCB P[]) {    for (int i = index; i < (\*n) - 1; i++) {      P[i] = P[i + 1];    }    (\*n)--;  }  int swapProcess(PCB \*P, PCB \*Q) {    PCB temp = \*P;    \*P = \*Q;    \*Q = temp;    return 0;  }  int partition(PCB P[], int low, int high, int iCriteria) {    int pivot = P[high].iArrival;    int i = low - 1;    for (int j = low; j < high; j++) {      switch (iCriteria) {        case SORT\_BY\_ARRIVAL:          if (P[j].iArrival <= pivot) {            i++;            swapProcess(&P[i], &P[j]);          }          break;        case SORT\_BY\_PID:          if (P[j].iPID <= pivot) {            i++;            swapProcess(&P[i], &P[j]);          }          break;        case SORT\_BY\_BURST:          if (P[j].iBurst <= pivot) {            i++;            swapProcess(&P[i], &P[j]);          }          break;        case SORT\_BY\_START:          if (P[j].iStart <= pivot) {            i++;            swapProcess(&P[i], &P[j]);          }          break;      }    }    swapProcess(&P[i + 1], &P[high]);    return i + 1;  }  void quickSort(PCB P[], int low, int high, int iCriteria) {      if (low < high) {          int pi = partition(P, low, high, iCriteria);          quickSort(P, low, pi - 1, iCriteria);          quickSort(P, pi + 1, high, iCriteria);      }  }  void calculateAWT(int n, PCB P[]) {      int sumWaitingTime = 0;      for (int i = 0; i < n; ++i) {          sumWaitingTime += P[i].iWaiting;      }      float avgWaitingTime = (float)sumWaitingTime / n;      printf("Average Waiting Time: %.2f\n", avgWaitingTime);  }  void calculateATaT(int n, PCB P[]) {      int sumTurnaroundTime = 0;      for (int i = 0; i < n; ++i) {          sumTurnaroundTime += P[i].iTaT;      }      float avgTurnaroundTime = (float)sumTurnaroundTime / n;      printf("Average Turnaround Time: %.2f\n", avgTurnaroundTime);  }  int main() {      PCB Input[10];      PCB ReadyQueue[10];      PCB TerminatedArray[10];      int iNumberOfProcess;      printf("Please input number of Process: ");      scanf("%d", &iNumberOfProcess);      int iRemain = iNumberOfProcess, iReady = 0, iTerminated = 0;      inputProcess(iNumberOfProcess, Input);      quickSort(Input, 0, iNumberOfProcess - 1, SORT\_BY\_ARRIVAL);      pushProcess(&iReady, ReadyQueue, Input[0]);      removeProcess(&iRemain, 0, Input);      ReadyQueue[0].iStart = ReadyQueue[0].iArrival;      ReadyQueue[0].iFinish = ReadyQueue[0].iStart + ReadyQueue[0].iBurst;      ReadyQueue[0].iResponse = ReadyQueue[0].iStart - ReadyQueue[0].iArrival;      ReadyQueue[0].iWaiting = ReadyQueue[0].iResponse;      ReadyQueue[0].iTaT = ReadyQueue[0].iFinish - ReadyQueue[0].iArrival;      printf("\nReady Queue: ");      printProcess(1, ReadyQueue);      while (iTerminated < iNumberOfProcess) {          while (iRemain > 0) {              if (Input[0].iArrival <= ReadyQueue[0].iFinish) {                  pushProcess(&iReady, ReadyQueue, Input[0]);                  removeProcess(&iRemain, 0, Input);                  continue;              }              else                  break;          }          if (iReady > 0) {              pushProcess(&iTerminated, TerminatedArray, ReadyQueue[0]);              removeProcess(&iReady, 0, ReadyQueue);                ReadyQueue[0].iStart = TerminatedArray[iTerminated - 1].iFinish;              ReadyQueue[0].iFinish = ReadyQueue[0].iStart + ReadyQueue[0].iBurst;              ReadyQueue[0].iResponse = ReadyQueue[0].iStart - ReadyQueue[0].iArrival;              ReadyQueue[0].iWaiting = ReadyQueue[0].iResponse;              ReadyQueue[0].iTaT = ReadyQueue[0].iFinish -ReadyQueue[0].iArrival;          }      }      printf("\n===== FCFS Scheduling =====\n");      exportGanttChart(iTerminated, TerminatedArray);      quickSort(TerminatedArray, 0, iTerminated - 1, SORT\_BY\_PID);      calculateAWT(iTerminated, TerminatedArray);      calculateATaT(iTerminated, TerminatedArray);        return 0;  } |

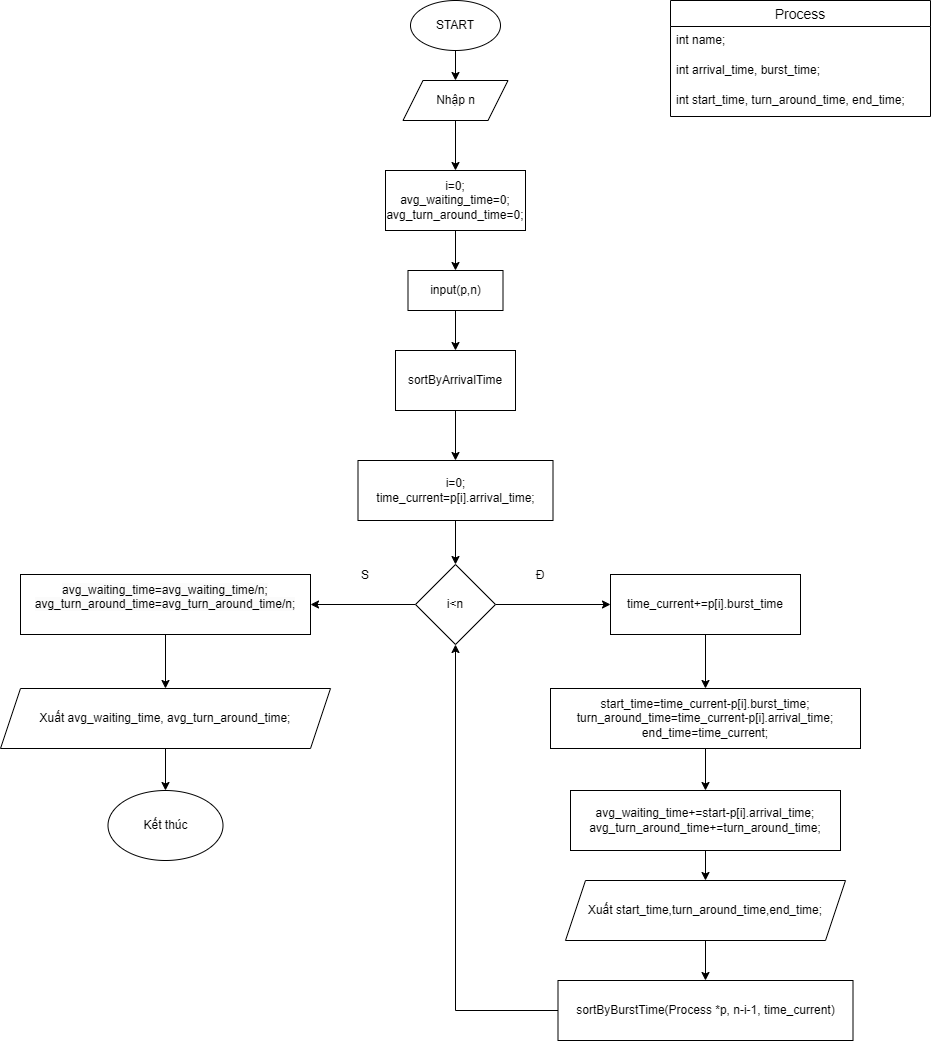
**Test case**

****

****

1. **SJF:**

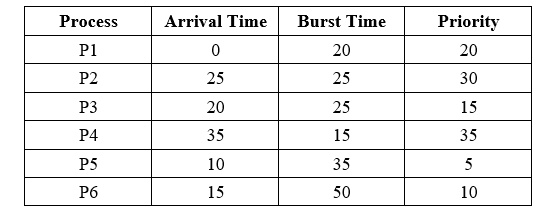
**Lưu đồ giải thuật**

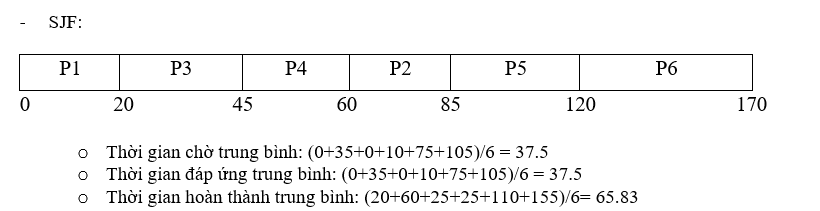


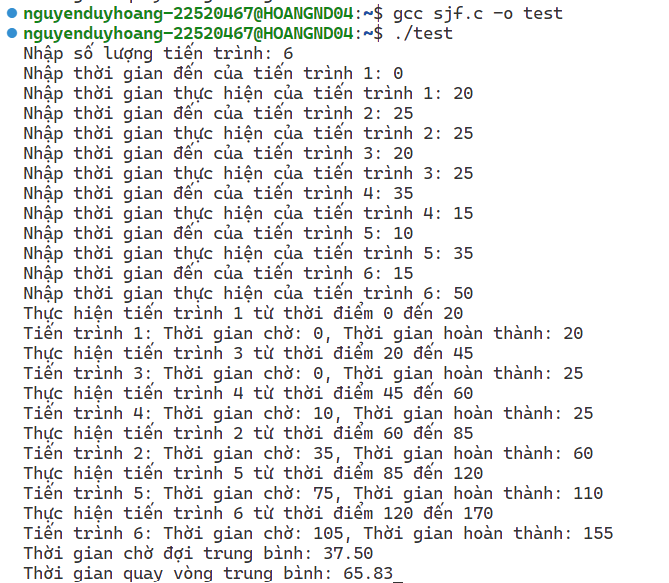
**Code**

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  // Cấu trúc biểu diễn tiến trình  typedef struct {      int iPID;      int arrival\_time;      int burst\_time;  } Process;  // Hàm so sánh cho sắp xếp theo thời gian đến  int compareByArrivalTime(const void \*a, const void \*b) {      return ((Process\*)a)->arrival\_time - ((Process\*)b)->arrival\_time;  }  // Hàm so sánh cho sắp xếp theo burst time  int compareByBurstTime(const void \*a, const void \*b) {      return ((Process\*)a)->burst\_time - ((Process\*)b)->burst\_time;  }  int main() {      int n;      printf("Nhập số lượng tiến trình: ");      scanf("%d", &n);      Process processes[n];      // Nhập thông tin cho từng tiến trình      for (int i = 0; i < n; i++) {          processes[i].iPID = i + 1;          printf("Nhập thời gian đến của tiến trình %d: ", i + 1);          scanf("%d", &processes[i].arrival\_time);          printf("Nhập thời gian thực hiện của tiến trình %d: ", i + 1);          scanf("%d", &processes[i].burst\_time);      }      // Sắp xếp tiến trình theo thời gian đến tăng dần      qsort(processes, n, sizeof(Process), compareByArrivalTime);      int time\_current = 0;      float avg\_waiting\_time = 0;      float avg\_turnaround\_time = 0;      for (int i = 0; i < n;) {          // Kiểm tra xem có tiến trình nào đến trong khoảng thời gian hiện tại không          if (processes[i].arrival\_time <= time\_current) {              int queue\_size = 0; // Kích thước của ready queue              int j = i;              // Tính kích thước của ready queue              while (j < n && processes[j].arrival\_time <= time\_current) {                  queue\_size++;                  j++;              }              // Sắp xếp các tiến trình trong ready queue theo burst time              qsort(processes + i, queue\_size, sizeof(Process), compareByBurstTime);              // Chọn tiến trình có burst time nhỏ nhất để thực hiện              Process selected\_process = processes[i];              // Thực hiện tiến trình              printf("Thực hiện tiến trình %d từ thời điểm %d đến %d\n",                     selected\_process.iPID, time\_current, time\_current + selected\_process.burst\_time);              // Cập nhật thời gian hiện tại              time\_current += selected\_process.burst\_time;              // Tính thời gian chờ đợi cho tiến trình              int waiting\_time = time\_current - selected\_process.arrival\_time - selected\_process.burst\_time;              // Cộng vào avg\_waiting\_time              avg\_waiting\_time += waiting\_time;              // Tính thời gian quay vòng cho tiến trình              int turnaround\_time = time\_current - selected\_process.arrival\_time;              // Cộng vào avg\_turnaround\_time              avg\_turnaround\_time += turnaround\_time;              // In thời gian chờ và thời gian hoàn thành của tiến trình              printf("Tiến trình %d: Thời gian chờ: %d, Thời gian hoàn thành: %d\n",                     selected\_process.iPID, waiting\_time, turnaround\_time);              // Tiến trình đã được thực hiện, tăng biến đếm              i++;          } else {              // Chưa có tiến trình nào đến, tăng thời gian hiện tại              time\_current++;          }      }      // Tính trung bình thời gian chờ đợi và thời gian quay vòng      avg\_waiting\_time /= n;      avg\_turnaround\_time /= n;      // In kết quả      printf("Thời gian chờ đợi trung bình: %.2f\n", avg\_waiting\_time);      printf("Thời gian quay vòng trung bình: %.2f\n", avg\_turnaround\_time);      return 0;  } |

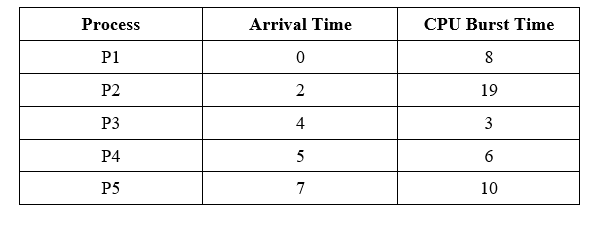
**Test case 1:**

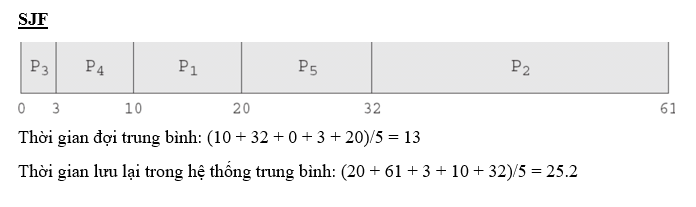
****

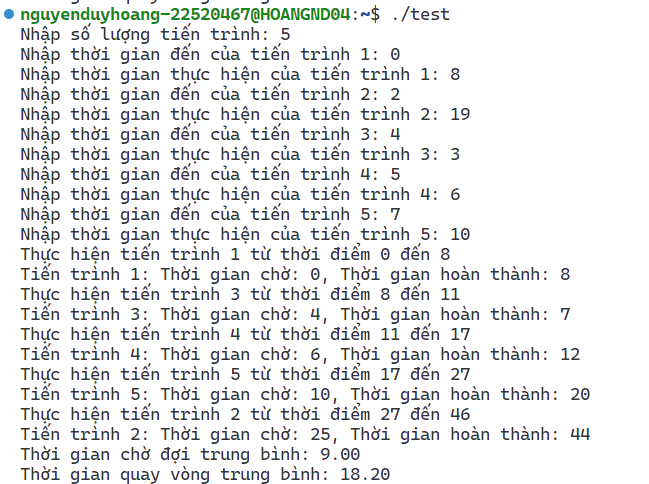
****

****

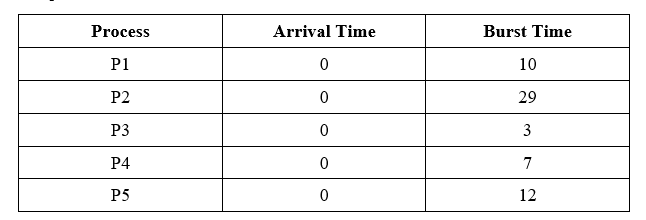
**Test case 2**

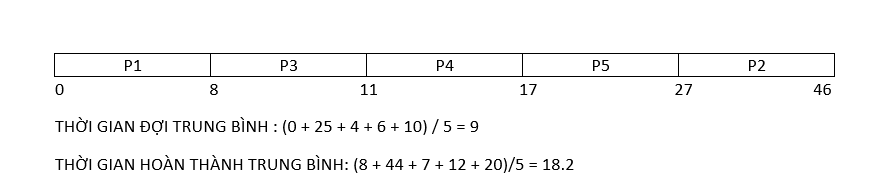
****

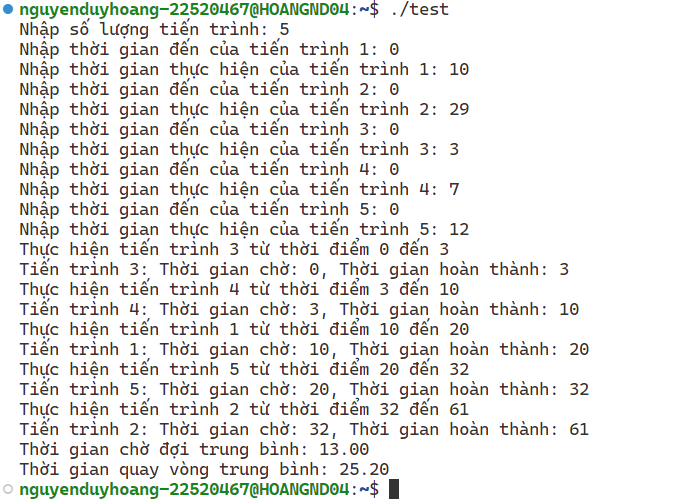
****

****

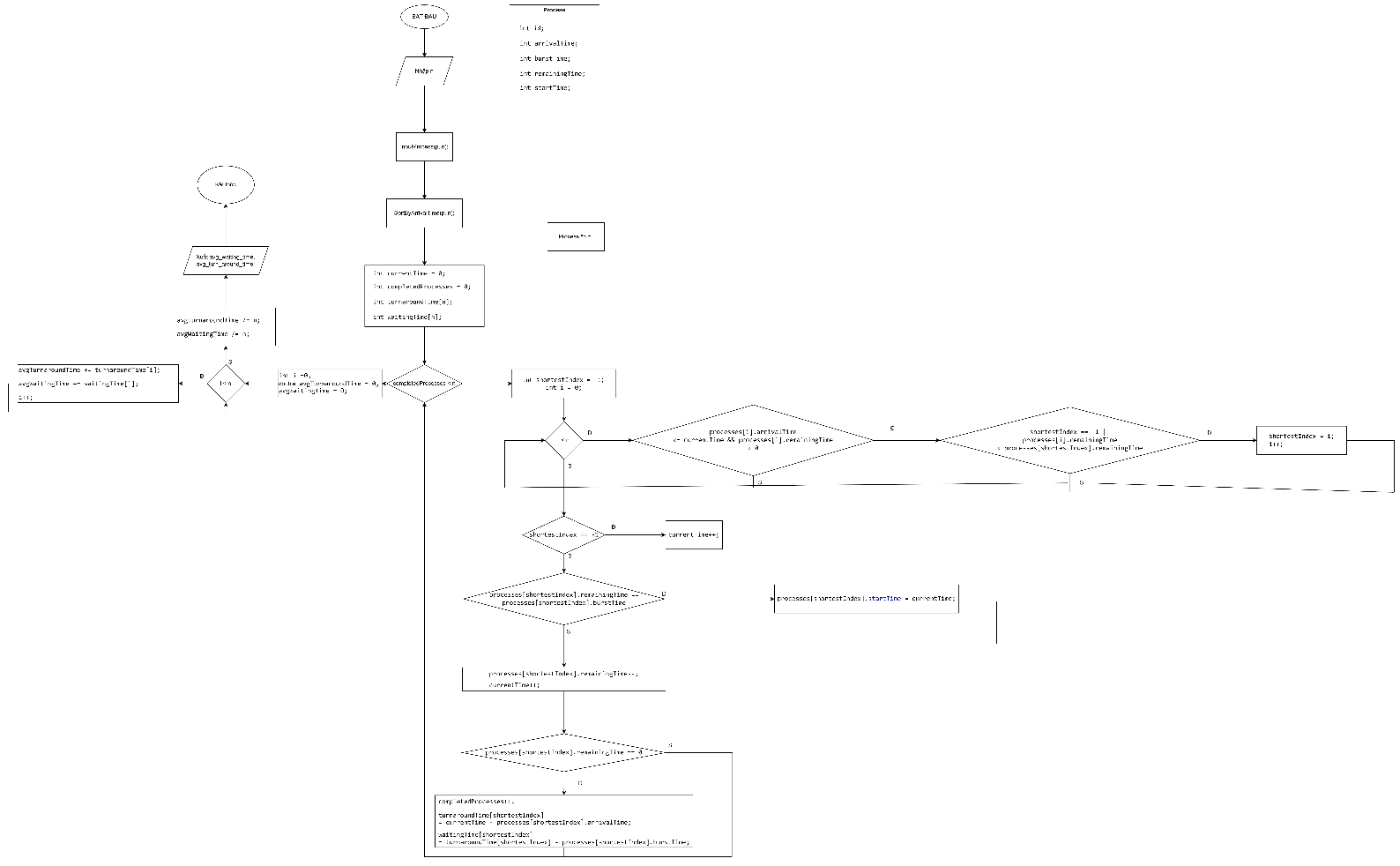
**Test case 3**

****

****

****

1. **SRTF:**

**Lưu đồ giải thuật**

**Code**

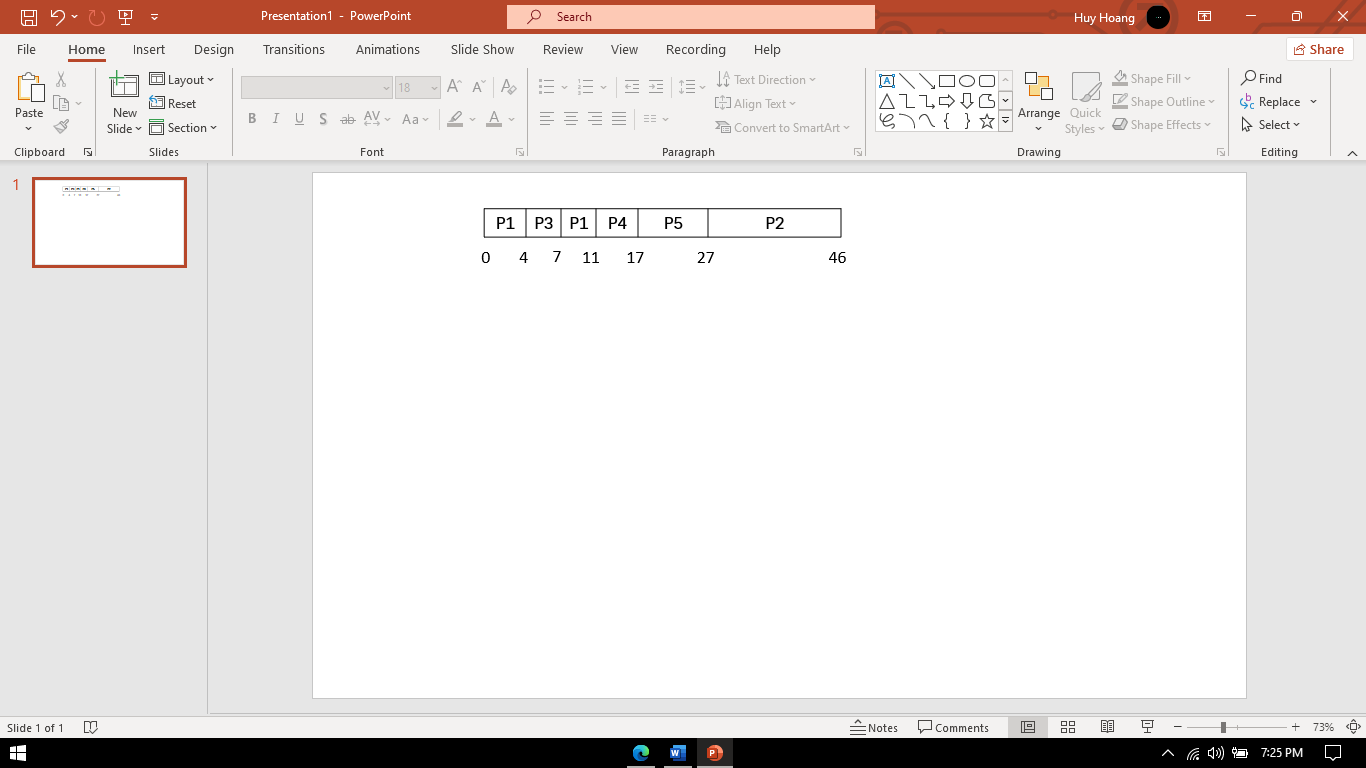
|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  struct Process {      int id;      int arrivalTime;      int burstTime;      int remainingTime;      int startTime;  // Thời gian bắt đầu thực hiện  };  void SRTF(struct Process \*processes, int n) {      // Sắp xếp các tiến trình theo thời gian đến      for (int i = 0; i < n - 1; ++i) {          for (int j = 0; j < n - i - 1; ++j) {              if (processes[j].arrivalTime > processes[j + 1].arrivalTime) {                  // Hoán đổi vị trí nếu thời gian đến của tiến trình sau lớn hơn tiến trình trước                  struct Process temp = processes[j];                  processes[j] = processes[j + 1];                  processes[j + 1] = temp;              }          }      }      int currentTime = 0;      int completedProcesses = 0;      int turnaroundTime[n];      int waitingTime[n];      while (completedProcesses < n) {          int shortestIndex = -1;          for (int i = 0; i < n; ++i) {              if (processes[i].arrivalTime <= currentTime && processes[i].remainingTime > 0) {                  if (shortestIndex == -1 || processes[i].remainingTime < processes[shortestIndex].remainingTime) {                      shortestIndex = i;                  }              }          }          if (shortestIndex == -1) {              currentTime++;          } else {              if (processes[shortestIndex].remainingTime == processes[shortestIndex].burstTime) {                  processes[shortestIndex].startTime = currentTime;              }              processes[shortestIndex].remainingTime--;              currentTime++;              if (processes[shortestIndex].remainingTime == 0) {                  completedProcesses++;                  turnaroundTime[shortestIndex] = currentTime - processes[shortestIndex].arrivalTime;                  waitingTime[shortestIndex] = turnaroundTime[shortestIndex] - processes[shortestIndex].burstTime;              }          }      }      // In kết quả      printf("Process\tTurnaround Time\tWaiting Time\n");      for (int i = 0; i < n; ++i) {          printf("%d\t%d\t\t%d\n", processes[i].id, turnaroundTime[i], waitingTime[i]);      }      // Tính toán và in thời gian chờ trung bình và thời gian hoàn thành trung bình      double avgTurnaroundTime = 0, avgWaitingTime = 0;      for (int i = 0; i < n; ++i) {          avgTurnaroundTime += turnaroundTime[i];          avgWaitingTime += waitingTime[i];      }      avgTurnaroundTime /= n;      avgWaitingTime /= n;      printf("\nAverage Turnaround Time: %.2f\n", avgTurnaroundTime);      printf("Average Waiting Time: %.2f\n", avgWaitingTime);  }  int main() {      // Nhập số lượng tiến trình      int n;      printf("Enter the number of processes: ");      scanf("%d", &n);      printf("---------------------------------------\n");      struct Process \*processes = (struct Process \*)malloc(n \* sizeof(struct Process));      // Nhập thông tin cho từng tiến trình      for (int i = 0; i < n; ++i) {          printf("Enter process id for process: ");          scanf("%d", &processes[i].id);          printf("Enter arrival time for process: ");          scanf("%d", &processes[i].arrivalTime);          printf("Enter burst time for process: ");          scanf("%d", &processes[i].burstTime);          processes[i].remainingTime = processes[i].burstTime;          printf("---------------------------------------\n");      }      // Chạy giải thuật SRTF      SRTF(processes, n);      free(processes);      return 0;  } |

**Test case 1 :**

|  |  |  |
| --- | --- | --- |
| **Process** | **Arrival Time** | **CPU Burst Time** |
| **P1** | **0** | **8** |
| **P2** | **2** | **19** |
| **P3** | **4** | **3** |
| **P4** | **5** | **6** |
| **P5** | **7** | **10** |

* *Giải tay*

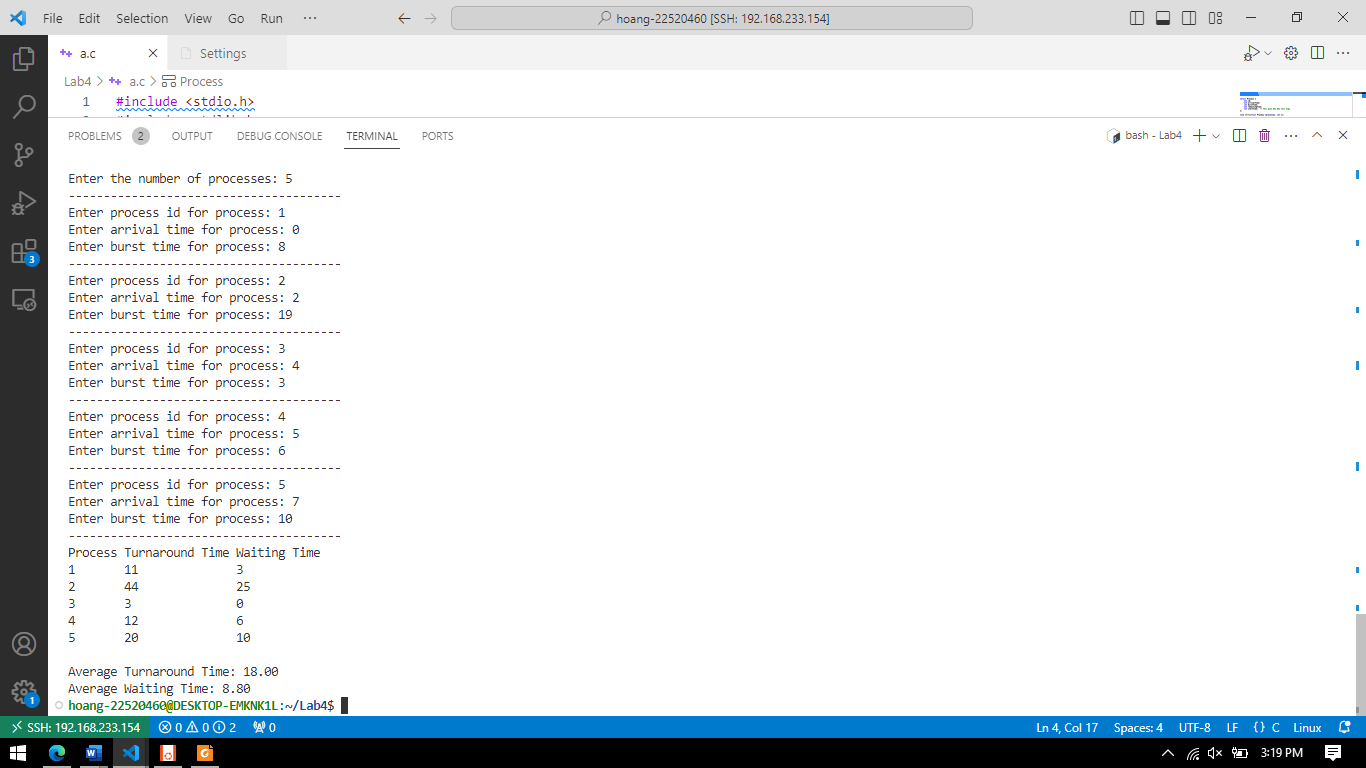
Giản đồ Gantt



Thời gian chờ trung bình : (3+ 25 + 0 + 6 + 10)/5 = 8,8

Thời gian lưu lại trong hệ thống: (11 + 44 + 3 + 12 + 20)/5 = 18

* *Chạy code*

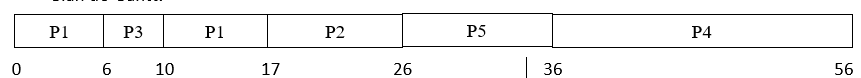


**Test case 2:**

|  |  |  |
| --- | --- | --- |
| **Process** | **Arrival Time** | **Burst Time** |
| **P1** | **0** | **13** |
| **P2** | **4** | **9** |
| **P3** | **6** | **4** |
| **P4** | **7** | **20** |
| **P5** | **12** | **10** |

* *Giải tay*

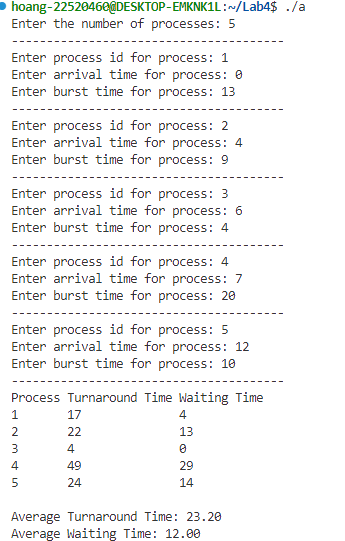
Giản đồ Gantt:



Thời gian đợi trung bình: (4 + 13 + 0 + 29 + 14)/5 = 12

Thời gian hoàn thành trung bình: (17 + 22 + 4 + 49 + 24)/5 = 23.2

* *Chạy code*

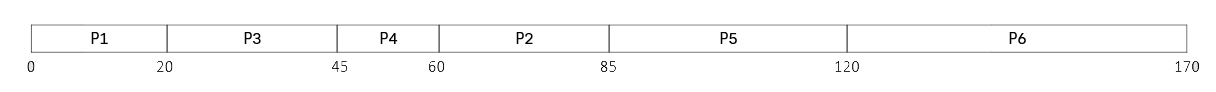
****

**Test case 3:**

|  |  |  |
| --- | --- | --- |
| **Process** | **Arrival Time** | **Burst Time** |
| **P1** | **0** | **20** |
| **P2** | **25** | **25** |
| **P3** | **20** | **25** |
| **P4** | **35** | **15** |
| **P5** | **10** | **35** |
| **P6** | **15** | **50** |

* *Giải tay*

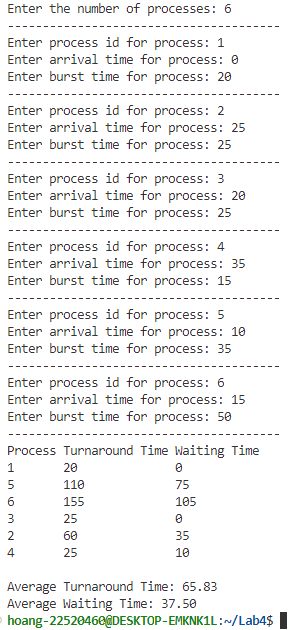
Giản đồ Gantt



Thời gian đợi trung bình: (0 + 35 + 0 + 10 + 75 + 105)/6 = 37,5

Thời gian hoàn thành trung bình: (20 + 60 + 25 + 25 + 110 + 155)/6 = 65,833

* *Chạy code*

****

1. **RR:**

**Lưu đồ giải thuật**

A diagram of a flowchart

Description automatically generated

**Code**

|  |
| --- |
| #include<stdio.h>  int pn[10], arr[10], arr1[10], bur[10], bur1[10], start[10], finish[10], tat[10], wt[10];  void hv(int a, int b)  {      int tmp = pn[a];      pn[a] = pn[b];      pn[b] = tmp;      tmp = arr[a];      arr[a] = arr[b];      arr[b] = tmp;      tmp = arr1[a];      arr1[a] = arr1[b];      arr1[b] = tmp;      tmp = bur[a];      bur[a] = bur[b];      bur[b] = tmp;      tmp = bur1[a];      bur1[a] = bur1[b];      bur1[b] = tmp;      tmp = start[a];      start[a] = start[b];      start[b] = tmp;      tmp = finish[a];      finish[a] = finish[b];      finish[b] = tmp;  }  void Sort(int b)  {      int i, j;      for(i = 0; i < b; i++)          for(j = i + 1; j <= b; j++)              if(arr1[i] > arr1[j])                  hv(i, j);              else if((arr1[i] == arr1[j]) && (bur1[i] > bur1[j]))                  hv(i, j);      for(i = b; i >= 1; i--)          if((arr1[i] == arr1[i-1]) && (arr[i] == arr1[i]) && (arr[i-1] == arr1[i - 1]))              hv(i, i - 1);  }  int main()  {      int totwt = 0, tottat = 0, i, n, j, QTime;      printf("Enter the number of processes: ");      scanf("%d", &n);      printf("Enter QTime: ");      scanf("%d", &QTime);      for (i = 0; i < n; i++)      {          printf("Enter the Process Name, Arrival Time, Burst Time: ");          scanf("%d%d%d", &pn[i], &arr[i], &bur[i]);          arr1[i] = arr[i];          bur1[i] = bur[i];      }      for(i = 0; i < n -1; i++)          for(j = i + 1; j < n; j++)          {              if(arr[i] > arr[j])                  hv(i, j);              else if((arr[i] == arr[j]) && (bur[i] < bur[j]))                      hv(i, j);          }      int PFinish = 0, time = 0;      while(PFinish < n)      {          while(arr1[0] > time)              time++;          if(arr[0] == arr1[0])              start[0] = time;          if(bur1[0] <= QTime)          {              time = time + bur1[0];              finish[0] = time;              hv(0, n-1-PFinish);              PFinish++;              Sort(n-1-PFinish);          }          else          {              time = time + QTime;              bur1[0] = bur1[0] - QTime;              arr1[0] = time;              Sort(n-1-PFinish);          }      }      printf("\nPName Arrtime Burtime Start Tat Finish Wt");      for(i = 0; i < n; i++)      {          tat[i] = finish[i] - arr[i];          wt[i] = tat[i] - bur[i];          printf("\nP%d\t%6d\t%6d\t%6d\t%6d\t%6d\t%6d", pn[i], arr[i], bur[i], start[i], tat[i], finish[i], wt[i]);          totwt += wt[i];          tottat += tat[i];      }      printf("\nAvgwt= %f\nAvgtat= %f\n", (float)totwt / n, (float)tottat / n);  return 0; |

**Test case 1**

RR(quantum time = 4)

A table with numbers and text

Description automatically generated

Kết quả giản đồ gantt:

A number of numbers and letters

Description automatically generated with medium confidence

Thời gian chờ trung bình: (14 + 10 + 21 + 10 + 18)/5 = 14,6

Thời gian chờ trung bình: (26 + 17 + 29 + 13 + 24)/5 = 21,8

A screen shot of a computer

Description automatically generated

**Test case 2**

RR (quantum time = 10)

|  |  |  |
| --- | --- | --- |
| Process | Arrival Time | Burst Time |
| P1 | 0 | 10 |
| P2 | 2 | 29 |
| P3 | 4 | 3 |
| P4 | 5 | 7 |
| P5 | 7 | 12 |

Kết quả giản đồ gantt:

Ảnh có chứa văn bản, Phông chữ, số, ảnh chụp màn hình

Mô tả được tạo tự động

Thời gian chờ trung bình: (0 + 30 + 16 + 18 + 33)/5 = 19,4

Thời gian chờ trung bình: (10 + 59 + 19 + 25 + 45)/5 = 31,6

A screen shot of a computer

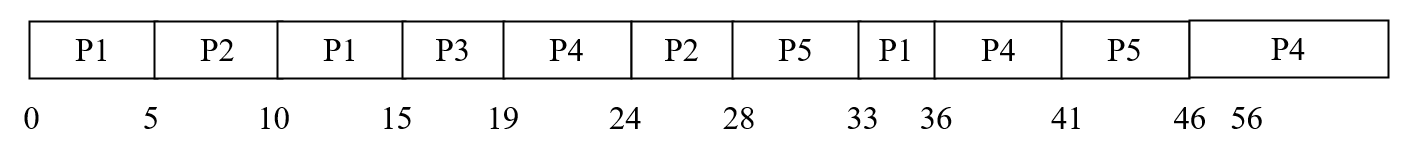
Description automatically generated

**Test case 3**

RR (quantum time = 5)

|  |  |  |
| --- | --- | --- |
| Process | Arrival Time | Burst Time |
| P1 | 0 | 13 |
| P2 | 4 | 9 |
| P3 | 6 | 4 |
| P4 | 7 | 20 |
| P5 | 12 | 10 |

Kết quả giản đồ gantt:



Thời gian đợi trung bình: ((5 + 18) + (1 + 14) + 9 + (12 + 12 + 5) + (16 + 8))/5 = 20

Thời gian hoàn thành trung bình: (36 + 24 + 13 + 49 + 34)/5 = 31.2

A screen shot of a computer

Description automatically generated