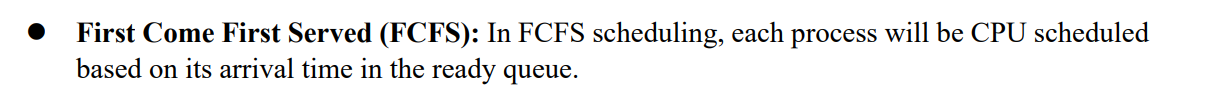
# Operating System Lab\_04

22CST 蒋云翔 2022102330

## Task1：

### Idea：

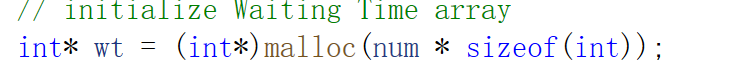
1. Teacher Bobo gives us a very **simple and convenient condition** in this lab, which is all the process **arrive at time 0**, significantly **simplify** the problem.



1. To store the corresponding burst time, waiting time, turnaround time. I decide to use malloc() function to create array to implement the requirements.

文本

中度可信度描述已自动生成

文本

中度可信度描述已自动生成

1. According to Teacher Bobo’s expected results, I thought the program should contain some constraints: limit the input numbers of processes(can not be too many!!!)

文本

中度可信度描述已自动生成

1. Actually this program still contains some bugs: ①My code is not very tidy and efficient enough ②The calculation of average waiting time and turnaround time better to be set as double type so that the decimal numbers can be shown.(These problem has been solved in Task2)

## Code structure

|  |
| --- |
| #include<stdio.h>  #include<stdlib.h>  int main()  {  int num;  printf("Plz enter total number of processes(more than 0 and less than 10:");  scanf("%d", &num);  printf("\n");  if (num < 0 || num > 10) {  printf("Invalid number of processes.\n");  return 1; // Invalid input!!!!Out!!  }  // dynamically malloc memory for processes  int\* bt = (int\*)malloc(num \* sizeof(int));  // Check whether malloc() successfully  if (bt == NULL) {  printf("Memory allocation failed.\n");  return 1; // If fail,then exit the program and print the reason for the failure  }  printf("Plz enter Process Burst Time\n");  for (int i = 0; i < num; i++)  {  printf("P[%d]:", i+1);  scanf("%d", &bt[i]);  }  // initialize Waiting Time array  int\* wt = (int\*)malloc(num \* sizeof(int));  int sum = 0;  wt[0] = 0;  for (int i = 1; i < num; i++)  {  sum += bt[i-1];  wt[i] = sum;  }  printf("\n");  // initialize Turnaround Time array  int\* tt = (int\*)malloc(num \* sizeof(int));  for (int i = 0; i < num; i++)  {  tt[i] = bt[i] + wt[i];  }  printf("Process\t\tBurst Time\tWaiting time\tTurnaround Time\n");  for (int i = 0; i < num; i++)  {  printf("P[%d]\t\t%d\t\t%d\t\t%d", i+1, bt[i], wt[i], tt[i]);  printf("\n");  }  printf("\n");  // Calculate Average Waiting Time  int AveWt = 0;  int time1 = 0;  for (int i = 0; i < num; i++)  {  time1 += wt[i];  AveWt = time1;  }  printf("Average Waiting Time:%d\n", AveWt/num);  // Calculate Average Turnaround Time  int AveTt = 0;  int time2 = 0;  for (int i = 0; i < num; i++)  {  time2 += tt[i];  AveTt = time2;  }  printf("Average Turnaround Time:%d", AveTt/num);  // Release the memory  free(bt);  free(wt);  free(tt);  return 0;  } |

## Execution results:

|  |
| --- |
|  |

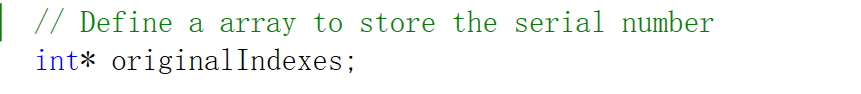
## Task2:

### Idea:

文本

描述已自动生成

1. When it refers to the word ‘shortest’, an idea bursts into my mind at the very moment that is **sorting** the array and the problem is solved perfectly.
2. Consider that we have to output the corresponding **serial number of the process**, I ultimately decide to use **Select sort** to implement it because I have to store it.(Have to malloc extra memory for an array to store the number)



## Code structure

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  // Define a array to store the serial number  int\* originalIndexes;  // Using Select Sort to implement SJF  void SortUsingSJF(int\* bt, int num) {  int i, j, minIndex, temp;  for (i = 0; i < num - 1; i++) {  minIndex = i;  for (j = i + 1; j < num; j++) {  if (bt[j] < bt[minIndex]) {  minIndex = j;  }  }  temp = bt[i];  bt[i] = bt[minIndex];  bt[minIndex] = temp;  temp = originalIndexes[i];  originalIndexes[i] = originalIndexes[minIndex];  originalIndexes[minIndex] = temp;  }  }  int main() {  int num;  printf("Enter number of processes: ");  scanf("%d", &num);  printf("\n");  if (num <= 0 || num > 10) {  printf("Invalid number of processes.\n");  return 1;  }  int\* bt = (int\*)malloc(num \* sizeof(int));  originalIndexes = (int\*)malloc(num \* sizeof(int));  int\* wt = (int\*)malloc(num \* sizeof(int));  int\* tt = (int\*)malloc(num \* sizeof(int));  if (bt == NULL || originalIndexes == NULL || wt == NULL || tt == NULL) {  printf("Memory allocation failed.\n");  return 1;  }  printf("Enter Burst Time for each process:\n");  for (int i = 0; i < num; i++) {  printf("p%d:", i + 1);  scanf("%d", &bt[i]);  originalIndexes[i] = i; // Initialize the original array  }  SortUsingSJF(bt, num);  // Initialize average time of wt and tt  int sum = 0;  for (int i = 0; i < num; i++) {  wt[i] = sum;  tt[i] = bt[i] + wt[i];  sum += bt[i];  }  printf("Process\tBurst Time\tWaiting Time\tTurnaround Time\n");  for (int i = 0; i < num; i++) {  printf("p%d\t\t%d\t\t%d\t\t%d\n", originalIndexes[i] + 1, bt[i], wt[i], tt[i]);  }  printf("\n");  // Calculate the average time  double AveWt = 0;  double AveTt = 0;  for (int i = 0; i < num; i++) {  AveWt += wt[i];  AveTt += tt[i];  }  printf("Average Waiting Time:%f\n", AveWt / num);  printf("Average Turnaround Time:%f\n", AveTt / num);  // Release memory  free(bt);  free(originalIndexes);  free(wt);  free(tt);  return 0;  } |

## Execution results:

|  |
| --- |
|  |