**گزارش آزمایش هشتم آزمایشگاه سیستم‌های عامل**

**اشکان شکیبا (۹۹۳۱۰۳۰)، علی هاشم‌پور (۹۹۳۱۰۸۲)**

الگوریتم FCFS:

یک الگوریتم بر پایه سیاست FIFO که not preemptive است.

#include<stdio.h>  
#include "time.h"  
  
struct process {  
 int pid;  
 int bt;  
 int wt;  
 int tt;  
};  
  
struct process p[20];  
  
int main() {  
 int n;  
 int i, j;  
 float average\_wt, average\_tt;  
 int sum\_wt = 0, sum\_tt = 0;  
 clock\_t startTime = clock();  
 printf("Number of process= ");  
 scanf("%d", &n);  
  
 printf("\nProcess Burst Time\n");  
  
 for (i = 0; i < n; i++) {  
 printf("\*P[%d]:", i + 1);  
 scanf("%d", &p[i].bt);  
 }  
  
 for (i = 0; i < n; i++) {  
 p[i].pid = i + 1;  
 }  
  
 p[0].wt = 0; //initial to zero  
  
 //calculate waiting time  
 for (i = 1; i < n; i++) {  
 p[i].wt = 0;  
 for (j = 0; j < i; j++)  
 p[i].wt = p[i].wt + p[j].bt;  
 }  
  
 printf("\nProcess\t |Burst Time |Waiting Time\t|Turnaround Time\n");  
 printf("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");  
  
 int count = 0;  
 //calculate turnaround time  
 for (i = 0; i < n; i++) {  
 p[i].tt = p[i].bt + p[i].wt;  
 sum\_wt = sum\_wt + p[i].wt;  
 sum\_tt = sum\_tt + p[i].tt;  
 printf("\n#P[%d]\t\t|%d\t\t\t |%d\t \t\t|\t%d", p[i].pid, p[i].bt, p[i].wt, p[i].tt);  
 count++;  
 }  
  
 average\_wt = sum\_wt / (float) count;  
 average\_tt = sum\_tt / (float) count;  
 printf("\n\n#Average Waiting Time= %0.2f", average\_wt);  
 printf("\n#Average Turnaround Time= %0.2f", average\_tt);  
 clock\_t endTime = clock();  
 printf("\n##Total time= %f\n", (double) (endTime - startTime) / CLOCKS\_PER\_SEC);  
 return 0;  
}

اگر تعداد پردازه‌ها را n در نظر بگیریم:

worst case time complexity: O(n2)

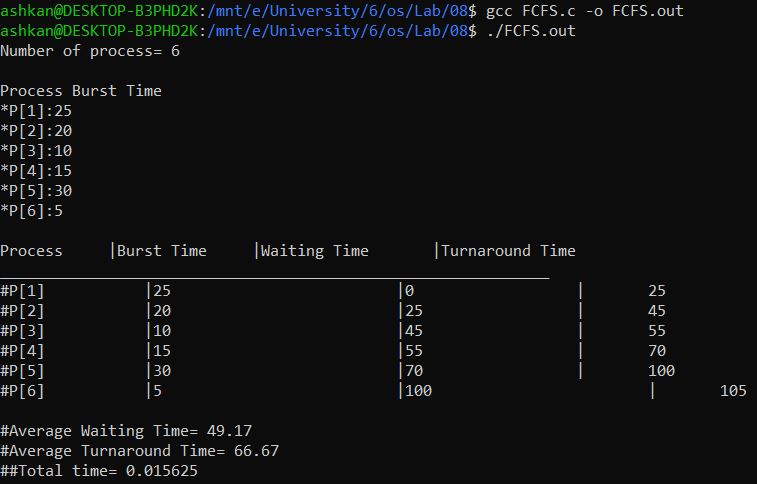
average time complexity: O(n2)

best case time complexity: θ(n)

space complexity = θ(1)

بعد از اجرای برنامه و تعیین burst time برنامه‌ها، می‌بینیم که زمان انتظار هر پردازه برابر با turnaround پردازه قبلی آن است. یعنی باید منتظر باشد تا پردازه قبلی کاملا به پایان رسیده و بعد اجرای خود را شروع کند.

خروجی برنامه:



الگوریتم SJF:

در این الگوریتم بین پردازه‌های با اولویت یکسان، پردازه با زمان مورد انتظار کمتر انتخاب و اجرا می‌شود. این الگوریتم نیز not preemptive است.

#include<stdio.h>  
#include "time.h"  
  
struct process {  
 int pid;  
 int bt;  
 int wt;  
 int tt;  
};  
  
struct process p[20];  
  
  
int main() {  
 int i, j;  
 int n;  
 int sum\_wt = 0, sum\_tt = 0;  
 int temp;  
 float average\_wt, average\_tt;  
 clock\_t startTime=clock();  
 printf("Number of process= ");  
 scanf("%d", &n);  
  
 printf("\nProcess Burst Time:\n");  
  
 for (i = 0; i < n; i++) {  
 printf("\*P[%d]:", i + 1);  
 scanf("%d", &p[i].bt);  
 }  
  
 for (i = 0; i < n; i++) {  
 p[i].pid = i+1;  
 }  
  
 //sort cpu burst time in ascending order  
 for (int i = 0; i < n - 1; i++) {  
 for (int j = 0; j < n - i - 1; j++) {  
 if (p[j].bt > p[j + 1].bt) {  
 temp = p[j].bt;  
 p[j].bt = p[j + 1].bt;  
 p[j + 1].bt = temp;  
  
 temp = p[j].pid;  
 p[j].pid = p[j + 1].pid;  
 p[j + 1].pid = temp;  
 }  
 }  
 }  
  
 p[0].wt = 0; //initial to zero  
  
 //calculate waiting time  
 for (i = 1; i < n; i++) {  
 p[i].wt = 0;  
 for (j = 0; j < i; j++) {  
 p[i].wt = p[i].wt + p[j].bt;  
 }  
 sum\_wt = sum\_wt + p[i].wt;  
 }  
  
  
 printf("\nProcess\t |Burst Time |Waiting Time\t|Turnaround Time\n");  
 printf("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");  
 for (i = 0; i < n; i++) {  
 p[i].tt = p[i].bt + p[i].wt;  
 sum\_tt = sum\_tt + p[i].tt;  
 printf("\n#P[%d]\t\t|%d\t\t\t |%d\t \t\t|\t%d", p[i].pid, p[i].bt, p[i].wt, p[i].tt);  
 }  
  
 average\_wt = (float) sum\_wt / n;  
 average\_tt = (float) sum\_tt / n;  
  
 printf("\n\n#Average Waiting Time= %0.2f", average\_wt);  
 printf("\n#Average Turnaround Time= %0.2f\n", average\_tt);  
 clock\_t endTime=clock();  
 printf("\n##Total time= %f\n",(double)(endTime-startTime)/CLOCKS\_PER\_SEC);  
}

اگر تعداد پردازه‌ها را n در نظر بگیریم:

worst case time complexity: O(n2)

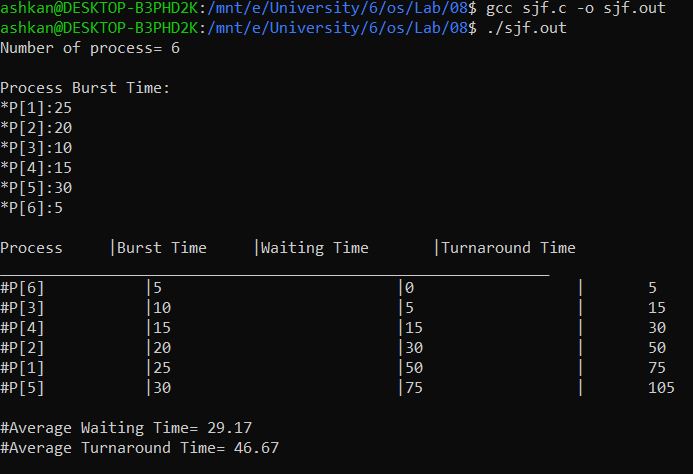
average time complexity: O(n2)

best case time complexity: θ(n)

space complexity = θ(1)

در این زمان‌بند بعد از اجرای برنامه و تعیین burst time برنامه‌ها، می‌بینیم که پردازه دارای بیشترین burst time آخر از همه اجرا شده و زمان‌بند با مرتب‌سازی پردازه‌ها، آنهایی که زمان کمتری نیاز دارند را اول اجرا می‌کند. این الگوریتم می‌تواند منجر به مشکل starvation شود.

خروجی برنامه:

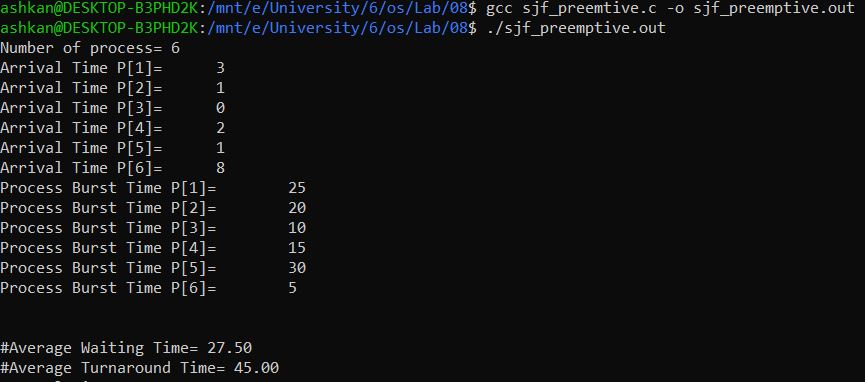


الگوریتم sjf preemptive:

تفاوت آن با الگوریتم قبل در این است که هر زمان در هنگام اجرای پردازه‌ای، پردازه جدیدی برسد که burst time آن از باقیمانده burst time پردازه فعلی کمتر باشد، اجرای پردازه کنونی متوقف شده و پردازه جدید جایگزین آن می‌شود. واضح است که این الگوریتم preemptive است.

#include<stdio.h>  
#include "time.h"  
  
struct process {  
 int pid;  
 int bt;  
 int wt;  
 int tt;  
 int ar;  
};  
  
struct process p[20];  
  
int main() {  
 int tmp[20];  
 int i;  
 int cnt = 0;  
 int n;  
 float sum\_wt = 0, sum\_tt = 0;  
 float average\_wt, average\_tt;  
 clock\_t startTime = clock();  
 printf("Number of process= ");  
 scanf("%d", &n);  
  
 for (i = 0; i < n; i++) {  
 printf("Arrival Time P[%d]= \t", i + 1);  
 scanf("%d", &p[i].ar);  
 }  
 for (i = 0; i < n; i++) {  
 printf("Process Burst Time P[%d]= \t", i + 1);  
 scanf("%d", &p[i].bt);  
 tmp[i] = p[i].bt;  
 }  
  
 for (i = 0; i < n; i++) {  
 p[i].pid = i + 1;  
 }  
  
 int j;  
 int index;  
 p[19].bt = 10000000;  
 int current\_time = 0;  
  
 while (cnt != n) {  
 index = 20-1;  
 for (i = 0; i < n; i++) {  
 if (p[i].ar <= current\_time && p[i].bt < p[index].bt) {  
 if (p[i].bt > 0) {  
 index = i;  
 }  
 }  
 }  
 p[index].bt--;  
  
 if (p[index].bt == 0) {  
 cnt++;  
 j = current\_time + 1;  
 sum\_wt += j - p[index].ar - tmp[index];  
 sum\_tt += j - p[index].ar;  
 }  
 current\_time++;  
 }  
  
 average\_wt = (float) sum\_wt / n;  
 average\_tt = (float) sum\_tt / n;  
 printf("\n\n#Average Waiting Time= %0.2f", average\_wt);  
 printf("\n#Average Turnaround Time= %0.2f", average\_tt);  
 clock\_t endTime = clock();  
 printf("\n##Total time= %f\n", (double) (endTime - startTime) / CLOCKS\_PER\_SEC);  
 return 0;  
}

خروجی برنامه:

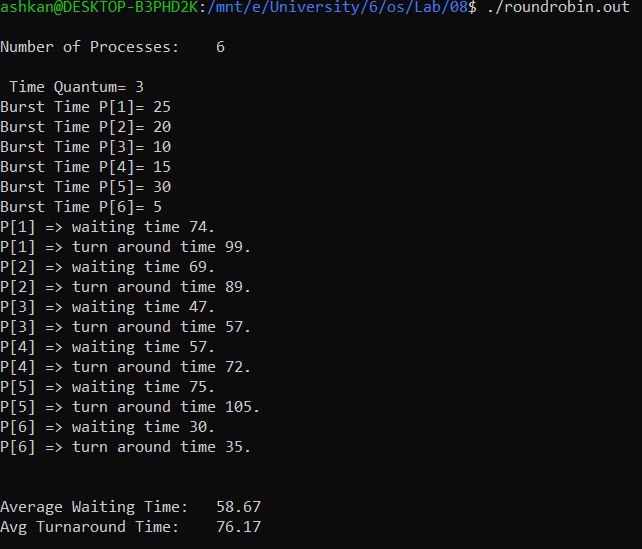


الگوریتم round robin:

الگوریتمی که به هر پردازه تنها به اندازه کوانتوم زمانی فرصت داده می‌شود و پس از پایان آن به صف انتظار فرستاده می‌شود تا دوباره نوبتش شود. این الگوریتم preemptive است.

#include<stdio.h>  
#include "time.h"  
  
struct process {  
 int pid;  
 int bt;  
 int wt;  
 int tt;  
 int ar;  
};  
struct process p[100];  
int l, r, TQ, n, cur;  
  
int main() {  
  
 int sum\_wt = 0, sum\_tt = 0;  
 int burstTime\_tmp[20];  
 float average\_wt, average\_tt;  
 clock\_t startTime = clock();  
 printf("\nNumber of Processes:\t");  
 scanf("%d", &n);  
 printf("\n Time Quantum= ");  
 scanf("%d", &TQ);  
  
 for (int i = 0; i < n; i++) {  
 p[i].pid = i + 1;  
 }  
 for (int i = 0; i < n; i++) {  
 printf("Burst Time P[%d]= ", p[i].pid);  
 scanf("%d", &p[i].bt);  
 burstTime\_tmp[r++] = i;  
 p[i].wt = -p[i].bt;  
 }  
  
 for (; l < r; l++) {  
 int i = burstTime\_tmp[l];  
 if (p[i].bt > TQ) {  
 cur += TQ;  
 p[i].bt -= TQ;  
 burstTime\_tmp[r++] = i;  
 } else {  
 cur += p[i].bt;  
 p[i].tt = cur;  
 p[i].wt += p[i].tt;  
 }  
 }  
  
 for (int i = 0; i < n; ++i) {  
 printf("P[%d] => waiting time %d.\n", p[i].pid, p[i].wt);  
 printf("P[%d] => turn around time %d.\n", p[i].pid, p[i].tt);  
 sum\_tt += p[i].tt;  
 sum\_wt += p[i].wt;  
 }  
  
  
 average\_wt = (float) sum\_wt / n;  
 average\_tt = (float) sum\_tt / n;  
 printf("\n\nAverage Waiting Time:\t%0.2f", average\_wt);  
 printf("\nAvg Turnaround Time:\t%0.2f\n", average\_tt);  
 clock\_t endTime = clock();  
 printf("\n##Total time= %f\n", (double) (endTime - startTime) / CLOCKS\_PER\_SEC);  
 return 0;  
}

خروجی برنامه:



الگوریتم priority:

این الگوریتم پردازه‌ها را با توجه به اولویت آنها انتخاب و اجرا می‌کند.

#include<stdio.h>  
#include "time.h"  
  
struct process {  
 int pid;  
 int bt;  
 int wt;  
 int tt;  
 int pr;  
};  
  
struct process p[20];  
  
  
void main() {  
 int temp, n, i;  
 clock\_t startTime = clock();  
 float average\_wt, average\_tt;  
 int sum\_wt = 0, sum\_tt = 0;  
 printf("Number of process= ");  
 scanf("%d", &n);  
  
 for (i = 0; i < n; i++) {  
 p[i].pid = i + 1;  
 }  
  
 for (i = 0; i < n; i++) {  
 printf("Burst Time P[%d]", p[i].pid);  
 scanf("%d", &p[i].bt);  
 printf("Priority P[%d]", p[i].pid);  
 scanf("%d", &p[i].pr);  
 }  
  
 //sort based on the priority  
 for (i = 0; i < n - 1; i++) {  
 for (int j = i + 1; j < n; j++) {  
 if (p[i].pr < p[j].pr) {  
 temp = p[i].pr;  
 p[i].pr = p[j].pr;  
 p[j].pr = temp;  
  
 temp = p[i].bt;  
 p[i].bt = p[j].bt;  
 p[j].bt = temp;  
  
 temp = p[i].pid;  
 p[i].pid = p[j].pid;  
 p[j].pid = temp;  
 }  
 }  
 }  
  
 p[0].wt = 0; //initialized into zero  
 p[0].tt = p[0].bt;  
 sum\_tt = p[0].tt;  
  
 for (i = 1; i < n; i++) {  
 p[i].wt = p[i - 1].tt;  
 sum\_wt = sum\_wt+ p[i].wt;  
 p[i].tt = p[i].wt + p[i].bt;  
 sum\_tt = sum\_tt+ p[i].tt;  
 }  
  
 printf("\nProcess\t |Burst Time |Waiting Time\t|Turnaround Time\n");  
 printf("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");  
  
 for (i = 0; i < n; i++) {  
 printf("\n#P[%d]\t\t|%d\t\t\t |%d\t \t\t|\t%d", p[i].pid, p[i].bt, p[i].wt, p[i].tt);  
 }  
 average\_wt = (float) sum\_wt / n;  
 average\_tt = (float) sum\_tt / n;  
 printf("\n Average Wait Time : %0.2f \n", average\_wt);  
 printf("\n Average Turn Around Time : %0.2f \n", average\_tt);  
 clock\_t endTime=clock();  
 printf("\n##Total time= %f\n",(double)(endTime-startTime)/CLOCKS\_PER\_SEC);  
}

خروجی برنامه:

