운영체제 HW1.

201613989 김진산

1. First In First Serve Algorithm

1-1) 현재 수정 전 코드에 대한 분석 내용을 기술하시오.

수정 전 FCFS(First Come First Server) 코드는 도착시간을 고려하지 않은 코드이며 프로세스들의대기 시간의 총 합과, 평균을 나타내며, 간트차트를 만들어 각각의 프로세스들의 순서, 실행 시간, 대기시간을 나타낸다.

변수는 프로세스의 개수 n, 실행시간 배열 b[], 대기시간 배열 w[], 프로세스들의 대기시간의 합 avg로 구성되어 있다. 배열 b, w의 크기가 10이므로 최대 프로세스의 개수는 10개이다.

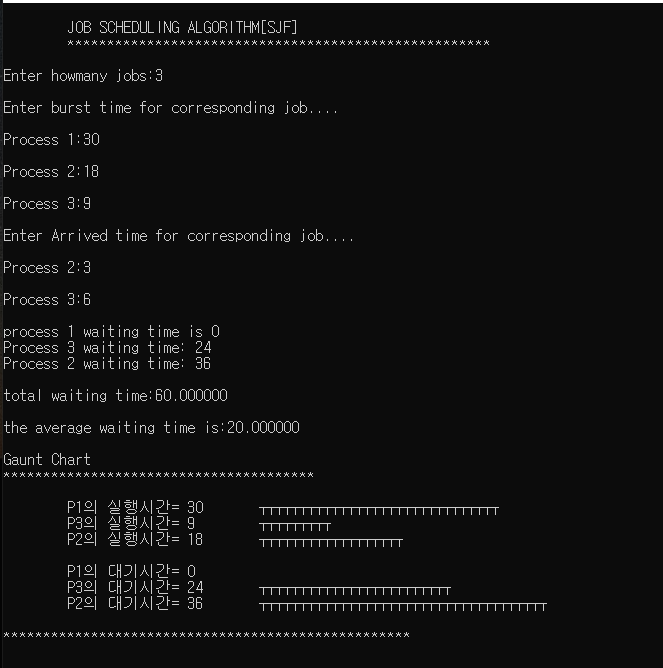
코드의 동작은 처음에 사용자로부터 프로세스의 개수 n을 입력받는다. 그후 n개의 프로세스들의 실행 시간을 입력받는다.

입력 받은 정보를 바탕으로 프로세스들의 대기시간을 측정한다. 첫번째 프로세스는 대기시간이 없으므로 w[0]=0으로 시작하며, 다음 프로세스들의 대기시간은 ‘이전 프로세스의 대기시간 + 이전 프로세스의 실행 시간’ 으로 구한다.

도착시간이 고려되지 않은 코드이므로 모든 프로세스의 도착시간은 0으로 정해져서 대기시간이 더 높게 측정된다. 그 결과로 평균 대기 시간도 높게 측정된다. 도착 시간이 모두 동일하므로 준비 큐에 어떤 프로세스가 먼저 도착했는지 판단할 수 없다는 문제 예상된다.

1-2) 이 코드를 이용하여 교재 그림 4-15와 동일한 결과가 출력되도록 코드를 수정하고, 실행 결과 화면을 보이시오.

기존에 주어진 코드의 간트차트는 프로세스들 간의 구분이 한눈에 들어오지 않아서 프로세스마다 한 줄로 표현했습니다.



1-3) 표를 이용하여, 수정전 코드에 대비하여 수정한 내용을 기술하시오.

|  |  |  |
| --- | --- | --- |
| 변경점 | 수정 전 코드 | 수정 후 코드 |
| Visual Studio에서 컴파일 오류 발생 | clrscr() | #include <windows.h>  system("cls"); |
| Visual Studio에서 컴파일 오류 발생 | delay(1000) | #include <stdlib.h>  Sleep(100); |
| Visual Studio에서 컴파일 오류 발생 | getch(); | \_getch(); |
| 도착 시간 입력 추가  (프로세스 0의 도착시간은 0으로 고정 – 항상 먼저 실행되는 프로세스로 가정) | int stime[10]; | int stime[10];  //도착시간 입력 (프로세스 0은 0으로 고정)  stime[0] = 0;  for (i = 1; i < n; i++)  {  printf("\nProcess %d:", i + 1);  scanf("%d", &stime[i]);  } |
| 도착 시간에 따른 우선순위 설정  (우선순위 설정에 따른 프로세스 번호 명시를 위해 a[] 배열 생성) | - | a[10];  //도착시간이 빠른것이 우선순위가 높음  for (i = 1; i < n; i++) {  for (j = i + 1; j < n; j++) {  if (stime[i] > stime[j])  {  t = b[i];  tt = a[i];  ttt = stime[i];  b[i] = b[j];  a[i] = a[j];  stime[i] = stime[j];  b[j] = t;  a[j] = tt;  stime[j] = ttt;  }  }  } |
| 도착 시간이 추가 됨에 따라 프로세스의 대기시간 변경  (대기시간 계산 시 도착시간 고려)  (준비 큐에 아직 프로세스가 도착하지 않은 경우가 있을 수 있다 판단하여, 후에 준비 큐에 들어온 프로세스의 대기시간은 0이라 판단..) | w[0]=0;  printf("\nprocess 1 waiting time is 0");  for(i=1;i<n;i++)  {  w[i]=b[i-1]+w[i-1];  printf("\nProcess %d waiting time: %d",i+1,w[i]);  avg+=w[i];  } | //처음 시작하는 프로세스는 대기시간이 없음  w[0] = 0; // w= waiting time  printf("\nprocess 1 waiting time is 0");  // 각각의 대기시간 구하기 //(대기시간 = 이전 프로세스 실행시간 + 이이전 프로세스 시작까지의 시간)  t = 0;  for (i = 1; i < n; i++)  {  if (t == 0) {  w[i] = b[i - 1] + w[i - 1];  }  else {  w[i] = b[i-1] + t;  }  //프로세스가 아직 도착하지 않은 경우 도착할 때까지 대기  if (w[i] < stime[i]) {  //도착과 동시에 실행되므로 대기시간 = 0;  t = stime[i];  w[i] = 0;  }  else {  t = w[i];  w[i] = w[i] - stime[i];  }  printf("\nProcess %d waiting time: %d", a[i], w[i]);  avg += w[i];  } |
| 간트 차트 모양 변경  (한눈에 보기 쉽도록) | for(i=0;i<n;i++)  {  printf("%d",b[i]);  for(j=1;j<=b[i];j++)  printf("%c",h);  }  printf("\n\t");  for(i=0;i<n;i++)  {  printf("%d",w[i]);  for(j=1;j<=w[i];j++)  printf("%c",h);  delay(1000);  } | // 실행시간 간트차트  for (i = 0; i < n; i++)  {  printf("P%d의 실행시간= %d \t", a[i], b[i]);  for (j = 1; j <= b[i]; j++)  printf("%c", h);  printf("\n\t");  Sleep(100);  }  printf("\n\t");  //대기시간 간트차트  for (i = 0; i < n; i++)  {  printf("P%d의 대기시간= %d\t", a[i],w[i]);  for (j = 1; j <= w[i]; j++)  printf("%c", h);  printf("\n\t");  Sleep(100);  } printf("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\t"); |

|  |
| --- |
| **전체코드**  #define \_CRT\_SECURE\_NO\_WARNINGS  #include <stdio.h>  #include <stdlib.h>  #include <conio.h>  #include <dos.h>  #include <windows.h>  void main()  {  int n; //프로세스 개수 10개로 제한되야함 (b, w의 크기가 10)  int b[10]; //실행 시간  int w[10]; //대기 시간  int a[10]; //프로세스 번호  int i, j; // for문 변수  int t, tt, ttt; // 도착시간에 따른 우선순위 변경을 위한 임시 변수  int h; //간트차트 표시 변수  int stime[10]; // 도착시간  float avg = 0; //모든 프로세스의 대기시간의 합  //메뉴 화면  system("cls"); //clear screen  printf("\n\tJOB SCHEDULING ALGORITHM[FCFS]");  printf("\n\t\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");  printf("\nEnter howmany jobs:");  scanf("%d", &n); // n= 일의 개수  //각각의 프로세스의 실행시간 입력  printf("\nEnter burst time for corresponding job....\n");  for (i = 0; i < n; i++)  {  printf("\nProcess %d:", i + 1);  scanf("%d", &b[i]);  a[i] = i+1;  }  //도착시간 입력 (프로세스 0은 0으로 고정)  printf("\nEnter Arrived time for corresponding job....\n");  stime[0] = 0;  for (i = 1; i < n; i++)  {  printf("\nProcess %d:", i + 1);  scanf("%d", &stime[i]);  }  //도착시간이 빠른것이 우선순위가 높음  for (i = 1; i < n; i++) {  for (j = i + 1; j < n; j++) {  if (stime[i] > stime[j])  {  t = b[i];  tt = a[i];  ttt = stime[i];  b[i] = b[j];  a[i] = a[j];  stime[i] = stime[j];  b[j] = t;  a[j] = tt;  stime[j] = ttt;  }  }  }  //처음 시작하는 프로세스는 대기시간이 없음  w[0] = 0; // w= waiting time  printf("\nprocess 1 waiting time is 0");  // 각각의 대기시간 구하기 //(대기시간 = 이전 프로세스 실행시간 + 이이전 프로세스 시작까지의 시간)  t = 0;  for (i = 1; i < n; i++)  {  if (t == 0) {  w[i] = b[i - 1] + w[i - 1];  }  else {  w[i] = b[i-1] + t;  }  //프로세스가 아직 도착하지 않은 경우 도착할때까지 대기  if (w[i] < stime[i]) {  //도착과 동시에 실행되므로 대기시간 = 0;  t = stime[i];  w[i] = 0;  }  else {  t = w[i];  w[i] = w[i] - stime[i];  }  printf("\nProcess %d waiting time: %d", a[i], w[i]);  avg += w[i];  }  //모든 프로세스의 대기시간 합 표기, 평균 대기시간 표기  printf("\n\ntotal waiting time:%f", avg);  printf("\n\nthe average waiting time is:%f\n", avg / n);  //간트차트 표시  printf("\nGauntChart\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\n\t");  h = 22; //(synchronous idle = 작은네모모양)  // 실행시간 간트차트  for (i = 0; i < n; i++)  {  printf("P%d의 실행시간= %d \t", a[i], b[i]);  for (j = 1; j <= b[i]; j++)  printf("%c", h);  printf("\n\t");  Sleep(100);  }  printf("\n\t");  //대기시간 간트차트  for (i = 0; i < n; i++)  {  printf("P%d의 대기시간= %d\t", a[i],w[i]);  for (j = 1; j <= w[i]; j++)  printf("%c", h);  printf("\n\t");  Sleep(100);  }  printf("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\t");  \_getch();  return;  } |

2. Shortest Job First Scheduling Algorithm

2-1) 현재 수정전 코드에 대한 분석 내용을 기술하시오.

수정 전 SJF(Shortest Job First Scheduling Algorithm) 코드는 도착시간을 고려하지 않은 코드이며 프로세스들의 대기 시간의 총 합과, 평균을 나타내며, 간트차트를 만들어 각각의 프로세스들의 순서, 실행 시간, 대기시간을 나타낸다.

변수는 프로세스의 개수 n, 실행시간 배열 b[], 대기시간 배열 w[], 프로세스 번호 a[], 프로세스들의 대기시간의 합 avg로 구성되어 있다. 배열 b, w의 크기가 10이므로 최대 프로세스의 개수는 10개이다.

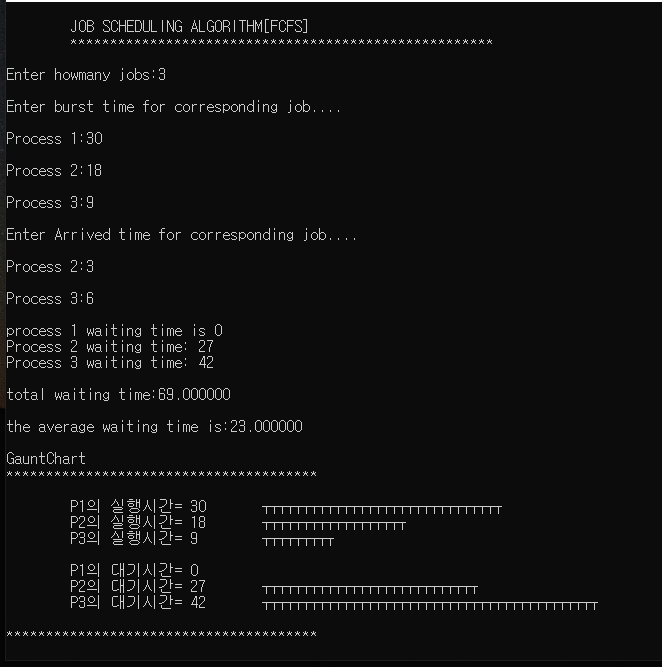
코드의 동작은 처음에 사용자로부터 프로세스의 개수 n을 입력받는다. 그후 n개의 프로세스들의 실행 시간을 입력받는다.

입력 받은 정보를 바탕으로 프로세스들 간의 우선순위를 비교한다. 실행 시간이 짧은 프로세스가 높은 우선순위를 실행시간이 긴 프로세스가 낮은 우선순위를 갖게 된다.

입력 받은 정보를 바탕으로 프로세스들의 대기시간을 측정한다. 첫번째 프로세스는 대기시간이 없으므로 w[0]=0으로 시작하며, 다음 프로세스들의 대기시간은 ‘이전 프로세스의 대기시간 + 이전 프로세스의 실행 시간’으로 구한다.

도착시간이 고려되지 않은 코드이므로 모든 프로세스의 도착시간은 0으로 정해져서 대기시간이 더 높게 측정된다. 그 결과로 평균 대기 시간도 높게 측정된다.

2-2) 이 코드를 이용하여 교재 그림 4-18과 동일한 결과가 출력되도록 코드를 수정하고, 실행 결과 화면을 보이시오.



2-3) 표를 이용하여, 수정전 코드에 대비하여 수정한 내용을 기술하시오

|  |  |  |
| --- | --- | --- |
| 변경점 | 수정 전 코드 | 수정 후 코드 |
| Visual Studio에서 컴파일 오류 발생 | clrscr() | #include <windows.h>  system("cls"); |
| Visual Studio에서 컴파일 오류 발생 | delay(1000) | #include <stdlib.h>  Sleep(100); |
| Visual Studio에서 컴파일 오류 발생 | getch(); | \_getch(); |
| 도착 시간 입력 추가  (프로세스 0의 도착시간은 0으로 고정 – 항상 먼저 실행되는 프로세스로 가정) | int stime[10]; | int stime[10];  도착시간 입력 (프로세스 0은 0으로 고정)  printf("\nEnter Arrived time for corresponding job....\n");  stime[0] = 0;  for (i = 1; i < n; i++)  {  printf("\nProcess %d:", i + 1);  scanf("%d", &stime[i]);  } |
| 실행 시간에 따른 우선순위변경  (도착시간 포함) | for(i=1;i<=n;i++)  for(j=i;j<=n;j++)  if(b[i]>b[j])  {  t=b[i]; tt=a[i];  b[i]=b[j]; a[i]=a[j];  b[j]=t; a[j]=tt;  } | for (i = 2; i <= n; i++) {  for (j = i + 1; j <= n; j++) {  if (b[i] > b[j]){  t = b[i];  tt = a[i];  ttt = stime[i];  b[i] = b[j];  a[i] = a[j];  stime[i] = stime[j];  b[j] = t;  a[j] = tt;  stime[j] = ttt;  }  }  } |
| 도착 시간이 추가 됨에 따라 프로세스의 대기시간 변경  (대기시간 계산 시 도착시간 고려)  (우선순위가 높은 프로세스가 준비 큐에 없을 수 있다 판단하여 준비 큐에 있는 우선순위가 높은 프로세스 먼저 실행하도록 판단.)  (준비 큐에 아직 프로세스가 도착하지 않은 경우가 있을 수 있다 판단하여, 후에 준비 큐에 들어온 프로세스의 대기시간은 0이라 판단..) | w[1]=0;  printf("\nprocess %d waiting time is 0",a[1]);  for(i=2;i<=n;i++)  {  w[i]=b[i-1]+w[i-1];  printf("\nProcess %d waiting time: %d",a[i],w[i]);  avg+=w[i];  } | //1번째 프로세스는 대기시간이 없음  w[1] = 0;  printf("\nprocess %d waiting time is 0", a[1]);  //대기시간 구하기 (대기시간 = 이전 프로세스 실행시간 + 이전 프로세스 시작까지의 시간)  for (i = 2; i <= n; i++){  if (tmp == 0) {  w[i] = b[i - 1] + w[i - 1];  }  else {  w[i] = b[i - 1] + tmp;  }  //우선순위가 가장 높은 일이 아직 도착하지 않았을 경우  if (stime[i] > w[i]) {  //도작한 프로세스중 가장 우선순위가 높은 프로세스 실행  for (j = i + 1; j <= n; j++) {  if (stime[j] <= w[i]) {  t = b[j];  tt = a[j];  ttt = stime[j];  for (int k = j - 1; k >= i; k--) {  b[k + 1] = b[k];  a[k + 1] = a[k];  stime[k + 1] = stime[k];  }  b[i] = t;  a[i] = tt;  stime[i] = ttt;  break;  }  }  }  //도착한 프로세스가 없을 경우 프로세스 도착까지 대기 후 우선순위가 높은 프로세스 실행  if (stime[i] > w[i]) {  tmp = stime[i];  //도착과 동시에 실행되므로 대기시간 = 0;  w[i] = 0;  }  else {  tmp = w[i];  w[i] = w[i] - stime[i];  }  printf("\nProcess %d waiting time: %d", a[i], w[i]);  avg += w[i];  } |
| 간트 차트 모양 변경  (한눈에 보기 쉽도록) | for(i=1;i<=n;i++)  {  printf("%d",b[i]);  for(j=1;j<=b[i];j++)  printf("%c",h);  }  printf("\n\t");  for(i=1;i<=n;i++)  {  printf("%d",w[i]);  for(j=1;j<=w[i];j++)  printf("%c",h);  delay(1000);  } | // 실행시간 간트차트  for (i = 1; i <= n; i++)  {  printf("P%d의 실행시간= %d \t", a[i], b[i]);  for (j = 1; j <= b[i]; j++)  printf("%c", h);  printf("\n\t");  Sleep(100);  }  printf("\n\t");  //대기시간 간트차트  for (i = 1; i <= n; i++)  {  printf("P%d의 대기시간= %d\t", a[i], w[i]);  for (j = 1; j <= w[i]; j++)  printf("%c", h);  printf("\n\t");  Sleep(100);  }  printf("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\t"); |

|  |
| --- |
| **전체 코드**  #define \_CRT\_SECURE\_NO\_WARNINGS  #include <stdio.h>  #include <stdlib.h>  #include <conio.h>  #include <dos.h>  #include <windows.h>  void main()  {  //변수선언  int n; //1~9 까지 프로세스의 개수  int b[10]; //실행 시간  int w[10]; //대기시간  int a[10]; // 프로세스 번호  int i, j; //for문 변수  int h; //간트차트 표시 변수  int t, tt, ttt, tmp=0; //임시 변수 (순서 변환)  int stime[10]; //도착시간  float avg = 0; // 모든 프로세스의 대기시간의 합  //메뉴화면  system("cls"); //clear screen  printf("\n\tJOB SCHEDULING ALGORITHM[SJF]");  printf("\n\t\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");  printf("\nEnter howmany jobs:");  scanf("%d", &n);  //각각의 프로세스의 실행시간 입력  printf("\nEnter burst time for corresponding job....\n");  for (i = 1; i <= n; i++)  {  printf("\nProcess %d:", i);  scanf("%d", &b[i]);  a[i] = i;  }  //도착시간 입력 (프로세스 1은 0으로 고정)  printf("\nEnter Arrived time for corresponding job....\n");  stime[1] = 0;  for (i = 2; i <= n; i++)  {  printf("\nProcess %d:", i);  scanf("%d", &stime[i]);  }  //우선순위 변경 (실행시간이 적을수록 우선순위가 높음)  for (i = 2; i <= n; i++) {  for (j = i + 1; j <= n; j++) {  if (b[i] > b[j]){  t = b[i];  tt = a[i];  ttt = stime[i];  b[i] = b[j];  a[i] = a[j];  stime[i] = stime[j];  b[j] = t;  a[j] = tt;  stime[j] = ttt;  }  }  }  //1번째 프로세스는 대기시간이 없음  w[1] = 0;  printf("\nprocess %d waiting time is 0", a[1]);  //대기시간 구하기 (대기시간 = 이전 프로세스 실행시간 + 이전 프로세스 시작까지의 시간)  for (i = 2; i <= n; i++){  if (tmp == 0) {  w[i] = b[i - 1] + w[i - 1];  }  else {  w[i] = b[i - 1] + tmp;  }  //우선순위가 가장 높은 일이 아직 도착하지 않았을 경우  if (stime[i] > w[i]) {  //도작한 프로세스중 가장 우선순위가 높은 프로세스 실행  for (j = i + 1; j <= n; j++) {  if (stime[j] <= w[i]) {  t = b[j];  tt = a[j];  ttt = stime[j];  for (int k = j - 1; k >= i; k--) {  b[k + 1] = b[k];  a[k + 1] = a[k];  stime[k + 1] = stime[k];  }  b[i] = t;  a[i] = tt;  stime[i] = ttt;  break;  }  }  }  //도착한 프로세스가 없을 경우 프로세스 도착까지 대기 후 우선순위가 높은 프로세스 실행  if (stime[i] > w[i]) {  tmp = stime[i];  //도착과 동시에 실행되므로 대기시간 = 0;  w[i] = 0;  }  else {  tmp = w[i];  w[i] = w[i] - stime[i];  }  printf("\nProcess %d waiting time: %d", a[i], w[i]);  avg += w[i];  }  //모든 프로세스의 대기시간합, 평균대기시간 표기  printf("\n\ntotal waiting time:%f", avg);  printf("\n\nthe average waiting time is:%f\n", avg / n);  //간트차트 표기  printf("\nGaunt Chart\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\n\t");  h = 22;  // 실행시간 간트차트  for (i = 1; i <= n; i++)  {  printf("P%d의 실행시간= %d \t", a[i], b[i]);  for (j = 1; j <= b[i]; j++)  printf("%c", h);  printf("\n\t");  Sleep(100);  }  printf("\n\t");  //대기시간 간트차트  for (i = 1; i <= n; i++)  {  printf("P%d의 대기시간= %d\t", a[i], w[i]);  for (j = 1; j <= w[i]; j++)  printf("%c", h);  printf("\n\t");  Sleep(100);  }  printf("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\t");  \_getch();  return;  } |