전자공학 프로그래밍 프로젝트1

전자공학과 201420653 최범준

**1. 구현된 함수의 기능 명세**

**2. 코드설명**

2-1) 주어진 함수설명

void sort(processStruct\* process[]) {

processStruct\* temp;

int i, j;

for (i = 4; i > 0; i--) {

for (j = 0; j < i; j++) {

if (process[j]->arrivalTime > process[j + 1]->arrivalTime) {

temp = process[j + 1];

process[j + 1] = process[j];

process[j] = temp;

}

}

}

printf("%d %d %d", process[0]->arrivalTime, process[1]->arrivalTime, process[2]->arrivalTime);

}

sort함수는 for문 내에 if문으로 process의 도착시간을 정렬하는 조건을 구현하여 1~5번 process를 순서대로 정렬해주는 기능을 수행.

void display(processStruct\* process[], int num, int time, int work) {

switch (work) {

case 0:

printf("시간: %d\b\n", time);

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

printf("현재 수행중인 process가 없습니다.\b\n");

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

for (int i = 0; i < 5; i++)

{

printf("Process : %s, Priority : %d, PerformTime : %d, ArrivalTime : %d, RemainingTime : %d \n", process[i]->name, process[i]->priority, process[i]->performTime, process[i]->arrivalTime, process[i]->leftoverTime);

}

break;

case 1:

printf("시간: %d\b\n", time);

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

printf("현재 수행중인 process의 이름은: %s\b\n", process[num]->name);

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

for (int i = 0; i < 5; i++)

{

printf("Process : %s, Priority : %d, PerformTime : %d, ArrivalTime : %d, RemainingTime : %d \n", process[i]->name, process[i]->priority, process[i]->performTime, process[i]->arrivalTime, process[i]->leftoverTime);

}

break;

case -1:

printf("시간: %d\b\n", time);

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

printf("모든 process가 종료되었습니다. \b\n");

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

break;

}

}

display함수는 case를 work인자의 -1,0,1 세가지 경우로 분류하여 각각 모든 process가 종료, 현재 수행중인 process가 없음, 현재 수행중인 process의 정보를 출력하여 주는 함수

void FCFS(processStruct\* process[]) {

processStruct\* temp;

int t = 0;

int check;

int work = 0;

int currentProcess = 0;

sort(process); //도착시간 순으로 배열을 정렬

while (1)

{

system("cls");

// process의 도착시간이 현재시간 보다 큰지 확인

if (process[currentProcess]->arrivalTime > t)

{

// 아직 도착하지 않은 상태이므로 해당 상태 출력

work = 0;

display(process, currentProcess, t, work);

}

else

{

// process가 도착한 경우

work = 1;

// 모든 process의 잔여 시간이 0인지 확인

check = 0;

for (int i = 0; i < 5; i++)

{

check += process[i]->leftoverTime;

}

// 모든 process가 종료된 경우

if (check <= 0)

{

work = -1;

display(process, currentProcess, t, work);

break;

}

// 현재 수행중인 process가 종료된 경우

if (process[currentProcess]->leftoverTime == 0)

{

//수행중이던 process가 마지막 process면 종료

if (currentProcess == 4)

{

work = -1;

display(process, currentProcess, t, work);

break;

}

currentProcess += 1;

continue;

}

// 현재 상태 출력 및 현재 수행중인 process의 잔여시간 1 감소

display(process, currentProcess, t, work);

process[currentProcess]->leftoverTime -= 1;

}

// 현재 시간 1 증가

t++;

Sleep(2000);

}

// 모든 process가 종료되면 malloc으로 할당해준 memory 할당해제

for (int i = 0; i < 5; i++)

{

free(process[i]);

}

}

FCFS함수는 우선 sort함수를 이용해 도착시간순으로 배열을 정렬한 뒤에 현재process의 도착시간이 현재시간보다 큰 경우 work=0값을 display함수에 넣어 수행중인 process가 없음을 출력하고 / 그 외에 도착한 process가 있는 경우에 모든 process의 잔여시간이 0인지 확인하고, 0인경우 work=-1값을 넣어 종료 문구를 출력/ 현재 수행중인 process의 잔여시간이 0이된경우 해당 process가 마지막 process인 경우 종료문구를 출력, 그 외의 경우에는 다음 process로 넘어가고 현재 상태 출력 및 process의 잔여시간 1 감소 및 현재시간 1 증가

void SJF(processStruct\* process[]) {

processStruct\* temp;

int t = 0;

int work = 0;

int currentProcess = 0;

int min = 100;

int check1;

int check2 = 0;

sort(process); //도착시간 순으로 배열을 정렬

while (1)

{

system("cls");

// process의 도착시간이 현재시간 보다 큰지 확인

if (process[currentProcess]->arrivalTime > t)

{

// 아직 도착하지 않은 상태이므로 해당 상태 출력

work = 0;

display(process, currentProcess, t, work);

}

else

{

// process가 도착한 경우

work = 1;

if (process[currentProcess]->leftoverTime == 0)

{

// 현재 수행중이던 process가 수행이 완료되었을 경우

// 수행시간이 가장 작은 process 찾기

for (int i = 0; i < 5; i++)

{

// 현재 도착하지 않았거나 수행이 완료된 process는 제외

if (process[i]->leftoverTime == 0 || process[i]->arrivalTime > t)

{

continue;

}

// 나머지 process 중

else if (process[i]->performTime < min)

{

// min보다 작은 값인 수행 시간을 가진 process를 현재 process로 변경

min = process[i]->performTime;

currentProcess = i;

}

}

min = 100;

// 위 과정을 커졌음에도 현재 process가 잔여 시간이 0인 경우, 아직 다음 proecess가 도착하지 않았거나 모든 process가 종료된 경우

if (process[currentProcess]->leftoverTime == 0)

{

// 모든 process의 잔여시간을 확인하여 모든 process가 종료되었는지 확인

check1 = 0;

for (int i = 0; i < 5; i++)

{

check1 += process[i]->leftoverTime;

}

if (check1 == 0)

{

// 모든 process가 종료되었다면 프로그램 종료

work = -1;

display(process, currentProcess, t, work);

break;

}

// 모든 process가 잔여시간이 0이거나 아직 도착한 process가 없는 경우

for (int i = 0; i < 5; i++)

{

if (process[i]->leftoverTime == 0 || process[i]->arrivalTime > t)

{

check2 += 1;

continue;

}

}

if (check2 == 5)

{

// 현재 process가 없다는 상태를 출력하고 현재 시간을 1증가

system("cls");

work = 0;

check2 = 0;

display(process, currentProcess, t, work);

t++;

Sleep(2000);

continue;

}

}

}

// 아직 잔여시간이 0이 아니고 process가 진행중인 경우 현재 상태 출력하고 잔여시간 1 감소

display(process, currentProcess, t, work);

process[currentProcess]->leftoverTime -= 1;

}

// 현재 시간 1증가

t++;

Sleep(2000);

}

// 모든 process가 종료되면 malloc으로 할당해준 메모리를 해제

for (int i = 0; i < 5; i++)

{

free(process[i]);

}

}

SJF 함수는 현재 도착한 함수가 없는경우 work=0을 넣어 상태 출력, 그 외의 경우에 잔여시간이 0이 아니고 process가 진행중인 경우 현재상태를 출력하고 잔여시간 1 감소 및 현재시간 1 증가 과정을 반복하는데 이 과정 중 현재진행 중인 process의 잔여시간이 0이 된 경우 나머지 process중 수행시간이 가장 작은 process를 찾아 우선수행(도착하지 않았거나 완료된 process는 제외) 또, 현재 진행중인 process가 없거나 모든 process가 종료되었는지 체크하여 해당하면 상태 출력

2-2) 작성한 함수설명

void PRIORITY(processStruct\* process[])

{

processStruct\* temp;

int max = 100;

int max1 = 0;

int t = 0;

int check1;

int check2 = 0;

int work = 0;

int currentProcess = 0;

sort(process); //도착시간 순으로 배열을 정렬

while (1)

{

system("cls");

// process의 도착시간이 현재시간 보다 큰지 확인

if (process[currentProcess]->arrivalTime > t)

{

// 아직 도착하지 않은 상태이므로 해당 상태 출력

work = 0;

display(process, currentProcess, t, work);

}

else

{

// process가 도착한 경우

work = 1;

// 우선순위가 가장 높은 process 찾기

for (int j = 0; j < 5; j++)

{

// 현재 도착하지 않았거나 수행이 완료된 process는 제외

if (process[j]->leftoverTime == 0 || process[j]->arrivalTime > t)

{

continue;

}

// 나머지 process 중

else if (process[j]->priority <= max)

{

// 우선순위가 같으면 먼저들어온 process를 현재 process로 변경

if (process[j]->priority == max)

{

currentProcess = max1;

}

// max보다 작은값인(우선순위가높은) 우선순위을 가진 process를 현재 process로 변경

else

{

max = process[j]->priority;

max1 = j;

currentProcess = j;

}

}

}

max = 100;

// 현재 process가 잔여 시간이 0인 경우에 아직 다음 proecess가 도착하지 않았거나모든 process가 종료된 경우

if (process[currentProcess]->leftoverTime == 0)

{

// 모든 process의 잔여시간을 확인하여 모든 process가 종료되었는지 확인

check1 = 0;

for (int i = 0; i < 5; i++)

{

check1 += process[i]->leftoverTime;

}

if (check1 == 0)

{

// 모든 process가 종료되었다면 프로그램 종료

work = -1;

display(process, currentProcess, t, work);

break;

}

// 모든 process가 잔여시간이 0이거나 아직 도착한 process가 없는 경우

for (int i = 0; i < 5; i++)

{

if (process[i]->leftoverTime == 0 || process[i]->arrivalTime > t)

{

check2 += 1;

continue;

}

}

if (check2 == 5)

{

// 현재 process가 없다는 상태를 출력하고 현재 시간을 1증가

system("cls");

work = 0;

check2 = 0;

display(process, currentProcess, t, work);

t++;

Sleep(2000);

continue;

}

}

// 현재 상태 출력 및 현재 수행중인 process의 잔여시간 1 감소

display(process, currentProcess, t, work);

process[currentProcess]->leftoverTime -= 1;

}

// 현재 시간 1 증가

t++;

Sleep(2000);

}

// 모든 process가 종료되면 malloc으로 할당해준 memory 할당해제

for (int i = 0; i < 5; i++)

{

free(process[i]);

}

}

Priority 함수는 도착한 process중 우선순위가 높은 process를 우선수행하는 함수로, 우선 도착한 함수가 없는경우 해당 상태를 출력하고, 그 외의 경우에 우선 순위가 높은 process를 수행하는 과정을 모든 process의 잔여시간이 0이 될 때 까지 반복 (상세설명은 주석첨부)

void SRT(processStruct\* process[])

{

processStruct\* temp;

int t = 0;

int work = 0;

int currentProcess = 0;

int lot = 100;

int lot1 = 0;

int check1;

int check2 = 0;

sort(process); //도착시간 순으로 배열을 정렬

while (1)

{

system("cls");

// process의 도착시간이 현재시간 보다 큰지 확인

if (process[currentProcess]->arrivalTime > t)

{

// 아직 도착하지 않은 상태이므로 해당 상태 출력

work = 0;

display(process, currentProcess, t, work);

}

else

{

// process가 도착한 경우

work = 1;

// 수행시간이 가장 작은 process 찾기

for (int k = 0; k < 5; k++)

{

// 현재 도착하지 않았거나 수행이 완료된 process는 제외

if (process[k]->leftoverTime == 0 || process[k]->arrivalTime > t)

{

continue;

}

// 나머지 process 중

else if (process[k]->leftoverTime <= lot)

{

//lot와 잔여시간이 같은 process가 있으면 우선순위 높은 것부터 수행

if (process[k]->leftoverTime == lot)

{

currentProcess = lot1;

}

// lot보다 작은 값인 잔여 시간을 가진 process를 현재 process로 변경

else

{

lot = process[k]->leftoverTime;

lot1 = k;

currentProcess = k;

}

}

}

lot = 100;

// 위 과정을 거쳤음에도 현재 process가 잔여 시간이 0인 경우, 아직 다음 proecess가 도착하지 않았거나 모든 process가 종료된 경우

if (process[currentProcess]->leftoverTime == 0)

{

// 모든 process의 잔여시간을 확인하여 모든 process가 종료되었는지 확인

check1 = 0;

for (int i = 0; i < 5; i++)

{

check1 += process[i]->leftoverTime;

}

if (check1 == 0)

{

// 모든 process가 종료되었다면 프로그램 종료

work = -1;

display(process, currentProcess, t, work);

break;

}

// 모든 process가 잔여시간이 0이거나 아직 도착한 process가 없는 경우

for (int i = 0; i < 5; i++)

{

if (process[i]->leftoverTime == 0 || process[i]->arrivalTime > t)

{

check2 += 1;

continue;

}

}

if (check2 == 5)

{

// 현재 process가 없다는 상태를 출력하고 현재 시간을 1증가

system("cls");

work = 0;

check2 = 0;

display(process, currentProcess, t, work);

t++;

Sleep(2000);

continue;

}

}

// 아직 잔여시간이 0이 아니고 process가 진행중인 경우 현재 상태 출력하고 잔여시간 1 감소

display(process, currentProcess, t, work);

process[currentProcess]->leftoverTime -= 1;

}

// 현재 시간 1증가

t++;

Sleep(2000);

}

// 모든 process가 종료되면 malloc으로 할당해준 메모리를 해제

for (int i = 0; i < 5; i++)

{

free(process[i]);

}

}

SRT함수는 SJF함수와 작동 방식이 비슷하나, 수행 중이던 process가 종료되지 않았더라도 도착한 함수 중 수행시간이 가장 짧은 process를 찾아 수행함. (상세설명 주석첨부)

void RR(processStruct\* process[])

{

processStruct\* temp;

int projectnum = 0;

int h;

int t = 0;

int check;

int work = 0;

int currentProcess = 0;

sort(process);

while (1)

{

system("cls");

// process의 도착시간이 현재시간 보다 큰지 확인

if (process[currentProcess]->arrivalTime > t)

{

// 아직 도착하지 않은 상태이므로 해당 상태 출력

work = 0;

display(process, currentProcess, t, work);

}

else

{

// process가 도착한 경우

work = 1;

// 모든 process의 잔여 시간이 0인지 확인

check = 0;

for (int i = 0; i < 5; i++)

{

check += process[i]->leftoverTime;

}

// 모든 process가 종료된 경우

if (check <= 0)

{

work = -1;

display(process, currentProcess, t, work);

break;

}

// 현재 상태 출력 및 현재 수행중인 process의 잔여시간 1 감소, 2초마다 다음 process로 넘어감

if (process[currentProcess]->leftoverTime > 0)

{

if (t % 2 == 0)

{

if (currentProcess == 4)

{

display(process, currentProcess, t, work);

process[currentProcess]->leftoverTime -= 1;

currentProcess = 0;

}

else

{

currentProcess += 1;

display(process, currentProcess, t, work);

process[currentProcess]->leftoverTime -= 1;

}

}

else

{

display(process, currentProcess, t, work);

process[currentProcess]->leftoverTime -= 1;

}

}

// 현재 수행중인 process가 종료되면 다음 process로 넘어가고 상태 출력

else if (process[currentProcess]->leftoverTime == 0)

{

currentProcess += 1;

display(process, currentProcess, t, work);

}

}

// 현재 시간 1 증가

t++;

Sleep(2000);

}

// 모든 process가 종료되면 malloc으로 할당해준 memory 할당해제

for (int i = 0; i < 5; i++)

{

free(process[i]);

}

}

RR함수는 2초마다 도착해있는 다음 process로 넘어가 수행하는 함수인데, 2초마다 넘어가는 과정은 구현했으나 다음 process로 넘어갔을 때 해당 process가 도착해있지 않은 상태이면 도착해있는 함수로 돌아가 수행하는 과정을 구현하지 못함.

**3. 결과화면에 대한 분석**

Priority)

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

Priority 함수는 현재 process가 종료되지 않았더라도 도착한 process 중 우선순위가 높은 process를 우선 수행하는 함수로 t=1에서 p1을 수행하다가 t=2에 p2가 도착하여 수행해서 t=5화면을 보면 p1의 잔여시간이 남았으나 p2를 수행중이고 p2의 잔여시간이 1로 남음 (이하동일)

SRT)

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

SRT함수는 SJF의 선점 방식으로 현재 수행중인 process가 종료되지 않았더라도 도착한 process중 잔여 수행시간이 적은 process를 우선수행 하고 잔여수행시간이 같은 경우 먼저 도착한 순으로 수행. t=1과 t=3 화면을 보면 t=2에 p2가 도착했고, t=2인 시점에 p1,p2 잔여 수행시간이 4로 같아 먼저 도착한 p1먼저 수행하고, t=9 화면을 보면 p2를 수행하고 있음. (이하동일)