**گزارش دستورکار هفتم آزمایشگاه سیستم­های عامل، قسمت دوم**

نگار موقتیان، 9831062

در این آزمایش می­خواهیم تعدادی پردازه را با توجه به زمان ورودی و burst time آن­ها زمانبندی کرده و waiting time و turnaround time را برای هر یک از پردازه­ها محاسبه کنیم. برای این کار از الگوریتم FCFS استفاده شده­است. در این الگوریتم هر پردازه­ای که زودتر وارد سیستم شود زودتر پردازش خواهد شد.

در برنامۀ نوشته شده یک struct به نام job تعریف شده­است که نمایندۀ هر یک از پردازه­های موجود در سیستم می­باشد. هر job یک arrival\_time یا زمان ورود، یک burst\_time یا مدت زمانی که می­خواهد از CPU استفاده کند و یک متغیر sliceable دارد که مشخص می­کند این کار قابل انجام شدن به صورت تکه تکه هست یا خیر.

از آن­جایی که در این برنامه از الگوریتم FCFS استفاده می­شود نیاز است که تابعی برای sort کردن پردازه­ها بر حسب زمان ورود آن­ها تعریف کنیم. این تابع یک تابع سادۀ sort است که از الگوریتم bubble sort استفاده می­کند.

در main برنامۀ نوشته شده ابتدا تعداد کاراها و پس از آن اطلاعات مربوط به آن­ها را دریافت کرده و در یک آرایه ذخیره­سازی می­کنیم. سپس این کارها را بر اساس زمان ورودشان مرتب می­کنیم.

حال تا زمانی که تمام کارها پردازش نشده­اند ادامه می­دهیم. متغیر time نشان­دهندۀ زمان فعلی، start\_time نشان­دهندۀ زمانی که پردازۀ فعلی کار خود را شروع کرده و متغیر curr نشان­دهندۀ این است که نوبت به اجرای کدام پردازه رسیده­. همچنین یک متغیر از نوع Boolean داریم که مشخص می­کند در حال حاضر پردازه­ای در حال اجرا هست یا خیر.

درون while برنامه ابتدا بررسی می­کنیم که پردازه­ای در حال اجرا هست یا خیر. اگر نبود بررسی می­کنیم که در زمان فعلی پردازۀ فعلی وارد سیستم شده­است یا خیر. اگر نشده بود باید همچنان صبر کنیم و اگر شده بود متغیر running را true کرده و start\_time را برابر با زمان فعلی قرار می­دهیم.

حال اگر پردازه­ای در حال اجرا بود و زمان اجرا آن تمام شده بود turnaround time (کل زمانی که پردازه به سیستم وارد شده تا کار آن به طور کامل تمام شده) و waiting time (زمانی که پردازه درون سیستم و صف ready بوده اما CPU به آن نرسیده) آن را چاپ کرده و به سراغ پردازۀ بعدی می­رویم. همچنین متغیر running را false می­کنیم تا در دور بعدی پردازۀ بعدی انتخاب شود.

در نهایت نیز یک واحد به زمان فعلی اضافه کرده و به ابتدای حلقه باز می­گردیم.

\* در صورتی که بخواهیم از قابلیت تکه تکه شدن پردازه­ها نیز استفاده کنیم می­توانیم از الگوریتم­ها قبضه­ای مانند Round Robin استفاده کنیم. در این الگوریتم کافیست زمانی که مدت زمان time quantum سپری شده­است بررسی کنیم که پردازه قابل تکه شدن هست یا خیر. اگر بود context switch را انجام می­دهیم و در غیر این صورت به اجرای پردازه تا به انتها ادامه می­دهیم.

#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

#include<stdbool.h>

struct job {

int arrival\_time;

int burst\_time;

int sliceable; // 0 -> cannot be sliced, 1 -> can be sliced

};

#define MAX\_PROCESS\_COUNT 20

int n;

struct job jobs[MAX\_PROCESS\_COUNT];

// swap two jobs by reference (used in sort algorithm)

void swap(struct job\* a, struct job\* b) {

struct job tmp = \*a;

\*a = \*b, \*b = tmp;

}

// sort processes according to their arriving time

void sort() {

for (int i=0; i<n-1; i++)

for (int j=0; j<n-i-1; j++)

if (jobs[j].arrival\_time > jobs[j+1].arrival\_time)

swap(&jobs[j], &jobs[j+1]);

}

int main() {

printf("Number of processes: ");

scanf("%d", &n);

printf("\nPlease enter 'arrival time' and 'burst time' corresponding to each process\nand determine if it 'can be sliced (1) or not (0)':\n");

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

scanf("%d%d%d", &jobs[i].arrival\_time, &jobs[i].burst\_time, &jobs[i].sliceable);

sort();

printf("\nSchedule:\n");

int time = 0, start\_time = 0, curr = 0;

bool running = false;

while (curr < n) {

if (!running && jobs[curr].arrival\_time <= time) { // select a new eligible job to run

running = true;

start\_time = time;

}

if (running && time - start\_time == jobs[curr].burst\_time) {

int tat = time - jobs[curr].arrival\_time;

int wt = start\_time - jobs[curr].arrival\_time;

printf("%d -> %d : Process %d (turnaround time = %d, waiting time = %d)\n", start\_time, time, ++curr, tat, wt);

running = false;

continue;

}

time++;

}

return 0;

}

خروجی این برنامه به ازای مثال داده شده مانند شکل زیر می­باشد.

