به نام خدا

ادامه‌ی گزارش آزمایش هشتم آزمایشگاه سیستم های عامل

زهرا رحیمی

شماره دانشجوئی: ۹۸۳۱۰۲۶

استاد آزمایشگاه: سرکار خانم حسینی

پاییز ۱۴۰۰

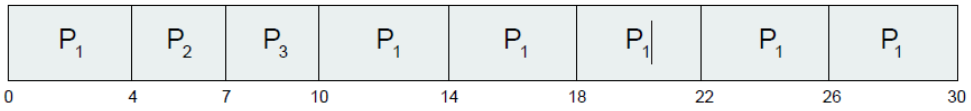
بخش چهارم: الگوریتم Round Robin:

```c
#include <stdio.h>
#include <math.h>
#define INT_MAX 2147483647

struct process{
    int pid;
    int bt;
    int wt, last_ex;
};

int main(){
    int process_num;
    scanf("%d", &process_num);
    struct process p[process_num];
    int burst_sum = 0;
    for (int i = 0; i<process_num; i++){
        int service_time;
        scanf("%d", &service_time);
        burst_sum += service_time;
        p[i].bt = service_time;
        p[i].pid = i;
        p[i].wt = 0;
        p[i].last_ex=0;
    }
    int q;
    scanf("%d", &q);

    int num = (int) ceil((double )burst_sum/q);
    int k = 0;
    int time = 0;

    for (int j = 0; j < num; j++) {
        if (k >= process_num)
            k = 0;
        if (p[k].bt <= q) {
            printf("Process %d has finished\n", p[k].pid);
            p[k].wt = p[k].wt + time - p[k].last_ex;
            p[k].last_ex = time + p[k].bt;
            p[j].bt = INT_MAX;
            time = p[k].last_ex;
        } else if (p[k].bt != INT_MAX){
            p[k].bt = p[k].bt - q;
            p[k].wt = p[k].wt + time - p[k].last_ex;
            printf("Process %d is executing\n", p[k].pid);
            p[k].last_ex = time + q;
            time = p[k].last_ex;
        } else
            j--;
        k++;
    }
    printf("\n");
    for (int l = 0; l < process_num; ++l) {
        printf("Process %d has waited %d seconds\n", p[l].pid, p[l].wt);
    }
    int avg_waiting = 0;
    for (int m = 0; m < process_num; ++m) {
        avg_waiting += p[m].wt;
    }
    printf("\nTotal Average Waiting is %.2f", (double )avg_waiting/process_num);
    return 0;
}
```

برای مثال می توانیم ۳ فرآیند با burst time های زیر به عنوان ورودی به این زمان بند بدهیم تا ببینیم چگونه آنها را زمان بندی می کند:

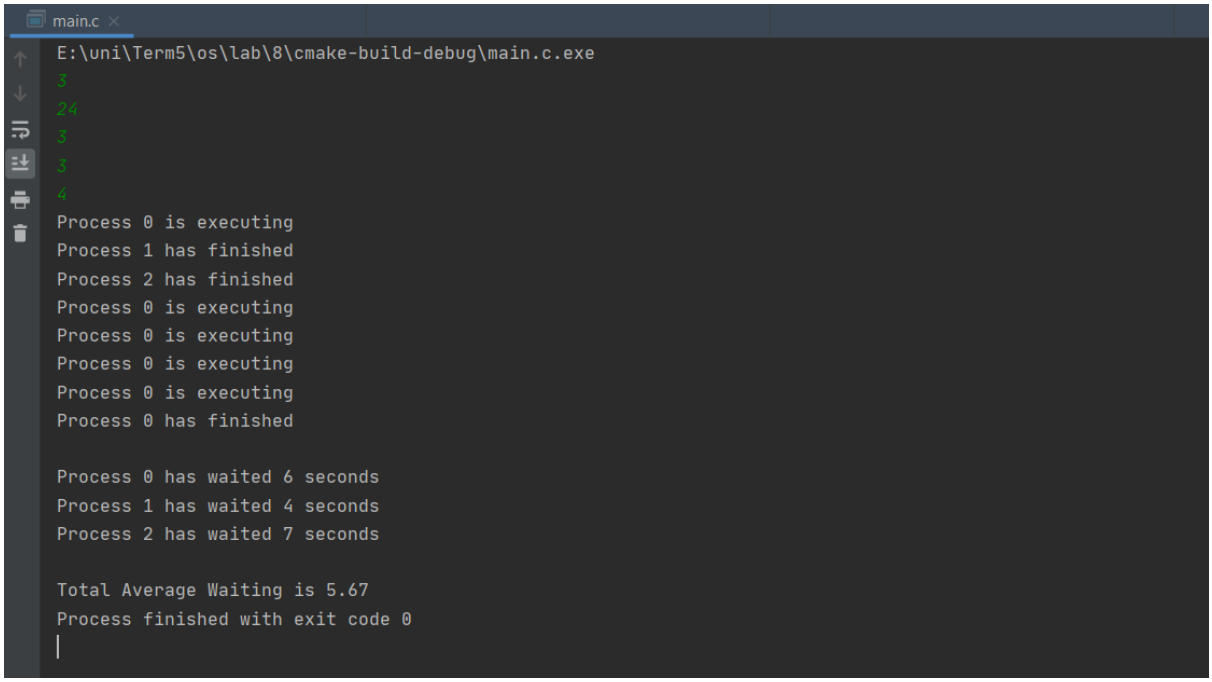
Process	Burst Time
P_1	24
P_2	3
P_3	3

The Gantt chart is:



Typically, higher average turnaround than SJF, but better ***response***
q should be large compared to context switch time
q usually 10ms to 100ms, context switch < 10 usec

خروجی به شکل زیر است و همان طور که انتظار می رفت اجرا شده است. برای محاسبه زمان انتظار در واقع زمان اجرای برنامه را از آخرین باری که اجرا شده است کم کرده و به زمان انتظار قبلی اضافه می کنیم.



بخش پنجم:

در این بخش نگاهی به مزایا و معایب همه الگوریتم ها می پردازیم:

الگوریتم	مزایا	معایب
FCFS	• سادگی و عدم پیچیدگی در پیاده سازی • عدم قحطی	اولویت با فرآیندهایی است که زودتر آمده اند و فرآیند های ته صف مجبور به انتظار طولانی هستند
SJF	افزایش تعداد فرآیند های انجام شده در زمان مشخص(Throughput)	• فرآیند های بزرگ مدت انتظار بیشتری دارند و به تدریج دچار قطحی می شوند. • زمان فرآیند ها از قبل برای CPU مشخص نیست
Priority based	انجام شدن سریع تر فرآیندهای بااولویت بالا	امکان به وجود آمدن قطحی برای فرآیند های با اولویت پایین
Round Robin	همه فرآیند ها به مقدار زمان مشخصی توسط CPU اجرا می شوند و قطحی پیش نمی آید	بسته به مقدار کوانتوم زمانی اگر کوچک باشد کارآیی CPU به علت سوییچ های متعدد پایین می آید و اگر بزرگ باشد مشکلات FCFS پیش می آید

با این توصیفات به طور کلی اگر اولویت اجرا شدن شان برای مان مهم است از الگوریتم Priority و اگر پیش روی شان برای مان اهمیت دارد Round Robin و اگر تعداد فرآیند هایی که در واحد زمان انجام می شوند مهم است SJF را استفاده کنیم. و به طور مشخص برای تعداد فرآیند های زیاد برای اینکه قطحی پیش نیاید و انجام شدن یا نشدن فرآیندی وابسته به اولویت یا کوتاهی انجام و یا در اول صف بودن نباشد، الگوریتم Round Robin مناسب تر به نظر می رسد ولی اگر هر کدام از مشخصه های بالا ملاک باشد باید متناسب با آن تصمیم گرفت. هم چنین همه معایب ذکر شده جاهایی است که نباید از این الگوریتم ها استفاده کنیم.